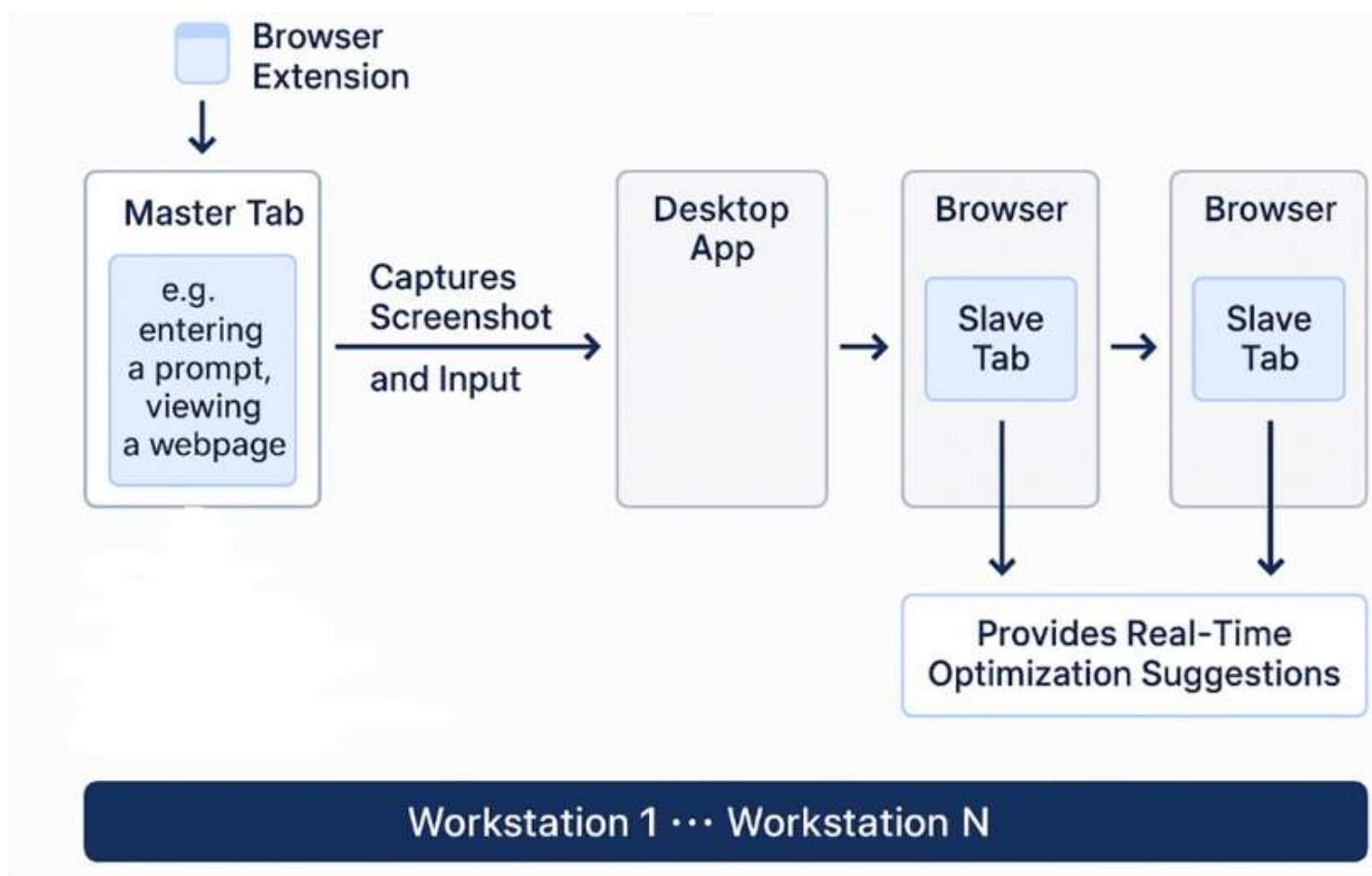
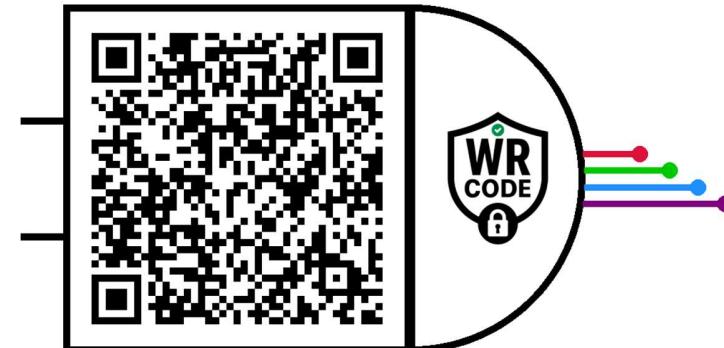


OpenGiraffe – A Browser-Based AI Agent Orchestration Framework For Real-Time Support



Paper by
optimando.ai



Abstract We introduce a novel source-available framework for browser-based AI agent orchestration, designed to deliver context-aware, real-time optimization suggestions that proactively assist users across digital activities. Developed under the direction of Oscar Schreyer and released under the **source-available** initiative of *optimando.ai*, a company specializing in AI automation for businesses —the system transforms the browser into a modular orchestration environment where distributed agents enhance user intent without requiring explicit commands.

WR CODE is built around a lightweight browser extension that links browser tabs with a local desktop orchestrator. Within this environment, users can assign roles to tabs—either as master interfaces (the primary point of interaction) or as helper agents that operate in parallel.

Helper agents are instantiated through templates, which may be injected automatically via WRCode scans, assembled from user-defined components, or provided as uploaded template files. Each agent can be supplied with relevant context, drawn either from WRCode-registered data, from local or organizational

vaults, or from personal notes and memos. The orchestrator also maintains a layered memory system, ranging from short-term session memory to long-term user or organizational memory, ensuring that reasoning can build upon prior interactions and adapt over time. This combination makes sure that outputs are grounded in precisely scoped information rather than generic knowledge.

To make workflows repeatable and scalable, the system allows entire sessions—including tab roles, orchestration logic, templates, context bindings, and memory states—to be saved and reloaded, eliminating the need for manual reconfiguration. The outputs of reasoning are displayed in structured grids, enabling users to compare, validate, and combine multiple agent responses side by side. In addition, the orchestrator provides pre-defined templates for intent recognition and real-time task support, allowing it to interpret user goals and surface relevant suggestions proactively. Agents require no coding: they are defined by plain-language instructions that make their behavior transparent and accessible.

This is a foundational design choice: all helper agents run inside the browser and use dedicated AI models—such as ChatGPT, Claude, DeepSeek, Gemini, Mistral, Llama, Grok, or autonomous systems like Google’s Project Mariner. By default these models are accessed via provider APIs, and each agent’s output is rendered in browser tabs within configurable display grids. When provider APIs are not available, and only where DOM manipulation is permitted under the relevant Terms of Service, data may alternatively be injected into provider web UIs or other software running in the browser.

In the default **WR CODE** architecture, AI helper and coordinator agents do not run inside tabs. They are defined in the on-prem orchestrator via templates (injected by **WRCode**, file upload, or user setup) that specify IDs, roles, endpoints, reasoning logic, and parameters. Agent outputs are rendered into configurable

display grids within helper tabs, while the master input tab context is preserved. When interfacing with web-only chatbot frontends, the orchestrator can manage real tabs to maintain context and simulate interaction. The orchestrator supports multiple profiles and modular agent configurations that can be toggled independently, enabling flexible activation of workflows or toolchains. Hosting the orchestration layer on-premise enhances privacy and data sovereignty and enables safe, community-driven sharing of reusable agent templates without exposing backend infrastructure. The traditional “one tab per agent” abstraction is therefore optional.

Adaptive Agent Architecture for Context-Aware Reasoning

Users configure AI agents in the system by defining behavior rules that guide how user input should be processed. A typical instruction might be:

“Interpret the user’s intent not only from the immediate input, but also from broader context, historical memory, and ongoing interaction patterns. Distinguish between short-term goals and long-term objectives. If helpful, generate up to five targeted follow-up questions or suggestions that either: (1) directly support the short-term goal, or (2) uncover smarter alternatives, missing steps, or overlooked opportunities that better align with the user’s long-term intent—even if these haven’t been explicitly requested.”

This high-level rule is executed by a chain of modular agents that work together to analyze, extend, and optimize user input dynamically.

Step 1: Input Coordination and Intent Expansion

All user input is first processed by an Input Coordinator Agent, which evaluates the complexity and intent of the request. It distinguishes between:

- Simple, factual queries → Direct response or light refinement
- Complex, strategic, or ambiguous inputs → Deeper inquiry and intent optimization

In the latter case, the agent generates follow-up prompts designed to clarify goals or surface smarter alternatives. These may include meta-level questions, such as:

"What additional questions could help uncover better options, reveal blind spots, or help the user reframe their original goal for a more robust outcome?"

"Could the user be aiming for an outcome they haven't articulated, possibly due to lack of technical vocabulary or system awareness?"

Unlike standard prompts, meta-level questions do not produce direct answers. Instead, they return new follow-up questions as output. This behavior triggers an internal feedback loop:

1. The meta-agent receives the original context and task framing.
2. It generates a set of secondary follow-up questions; each aimed at improving strategic alignment or surfacing hidden insights.

3. These secondary questions are routed to new Helper Agents, just like primary ones.
4. Only once those agents return their responses does the system proceed to synthesis and output.

This recursive mechanism allows the system to reason not only about *what* the user asked, but *whether they're asking the right questions*—turning the orchestration into a continuously optimizing cycle.

Step 2: Distributed Parallel Reasoning via Helper Agents

Each follow-up question is then delegated to a specialized Helper Agent, which processes it independently. These agents work in parallel, using tailored logic and data access scopes.

There are two execution modes:

1. **API-Driven Orchestration (Default)**

Agents are defined in the on-prem orchestrator via templates (WRCode, file upload, or user setup). API credentials are **managed and metered by WRCode.org** at the account level, or run in **BYOK** mode where users provide keys locally; the orchestrator itself is local-only and not a cloud service. Agents do not run in tabs; their outputs are rendered into configurable **display grids** within helper tabs, while the master input tab context is preserved.

2. Web-UI Automation (ToS-Gated Fallback)

For services that expose only a browser UI (no API), the orchestrator manages real tabs to maintain context and simulate interaction—strictly where DOM automation is permitted by the provider’s ToS.

In this mode only, users must supply and manage their own site credentials; setup is service-specific and manual, intended for edge-case custom integrations. Agent logic remains in the orchestrator; tabs are used purely as UI surfaces.

Regardless of the mode, Helper Agents may utilize:

Context, Memory & Tooling (Rewritten)

- **Public source of truth (WRCode.org):**

Public WRCode templates—**context packages, template maps, AI instructions**—are **published and embedded on WRCode.org** (not locally). They are **tamper-proof, auditable, verifiable, and versioned**. During inference, the local orchestrator performs **real-time retrieval** from WRCode.org’s index (with optional transient caching or offline bundles).

- **Archive & offline use:**

Users may **manually browse and download** signed/hashed releases from the **WRCode.org archive** for offline or air-gapped setups; integrity is verified via checksums/signatures.

- **Group-scoped publisher context (non-sensitive):**

Publishers can upload **non-sensitive** materials to WRCode.org with **restricted access** (e.g., family/team). These assets remain auditable and verifiable while honoring declared scopes.

- **Internal-only deployments (local embedding):**
If WRCode is used **only internally**, publishers **embed and index context locally** on their own infrastructure (PC orchestrator, smartphone, or other devices). Realistic defaults: **~100 pages on smartphones** and **~5,000 pages on PCs** (configurable). **All materials are pre-embedded up front for performance**; query-time retrieval operates on these precomputed embeddings (no on-the-fly embedding).
- **Sensitive user context & memory (device-side):**
Sensitive materials are **kept local** and embedded on-device in an **embedded vector DB**. Scopes/TTL, purge, and audit options are enforced locally; **nothing is uploaded to WRCode.org**.
- **User modifications (no auto-overwrite):**
If AI instructions or template fields are **edited locally**, they are marked **modified**. Subsequent scans/updates from WRCode.org **do not overwrite** automatically; the user is prompted to **approve overwrite or merge**. Until approved, the **local modified version remains in effect**.
- **Automation & models:**
The orchestrator includes **built-in automation tools** on PC and mobile and **bundled small LLMs** (including on mobile). Users can embed **more powerful stacks (e.g., n8n)** or route to larger/external models via API.
- **Filters, prompts, tools:**
Helpers can apply **context-specific filters** (e.g., “safety documents only”), **agent-specific prompt engineering**, and **tool access policies**.

- **Credentials & billing alignment:**

For API-driven use, credentials are **managed/metered by WRCode.org** at the account level, or run in **BYOK** mode with user-supplied keys (per workspace/agent). The **orchestrator is local-only** and is **not a cloud service**.

This modular parallelism enables domain-specific specialization, such as pricing analysis, compliance verification, or user behavior prediction.

Step 3: Supervised Synthesis and Recursive Optimization

Once all Helper Agents return their results, the outputs are passed to a Supervising Agent, which:

- Merges, compares, and synthesizes all responses
- Detects redundancies, contradictions, or knowledge gaps
- Identifies emergent patterns and opportunities
- Triggers recursive follow-up actions if deeper inquiry is warranted

This Supervising Agent forms the core of the system's intelligent feedback loop—enhancing the user's original request based on learned context, not just answering it.

 Step 4: Output Coordination and Display Logic

Final results are routed to the Output Coordinator Agent, which governs how information is presented to the user through flexible, screen-aware display slots. These slots are implemented into display grids in browser tabs or as dynamic flexbox containers that scale fluidly to fill available screen space across single or multiple monitors. The layout is fully customizable, allowing users to define the size, position, and behavior of each slot based on their workflow needs. To further enhance usability, the primary interaction layer—hosted in the master browser tab—can be looped through and snapped into the same grid. This is achieved by cloning the master tab and integrating it seamlessly into the layout, preserving a clear separation between orchestration controls and content delivery without sacrificing screen real estate. The Output Coordinator handles:

- Slot allocation, ensuring results are displayed in areas that reflect their importance and relevance
- Smooth transitions and display timing, including fade-ins, freezes, and content replacement logic
- Responsiveness and contextual awareness, adapting presentation based on the evolving task flow

- Event-Driven Refresh – New queries, memory updates, or evolving user goals can trigger partial or full re-rendering of display slots
- Topic Transitions – When a user switches topics, previous outputs may fade or be archived, while a “Back” button allows recovery of recent insights
- Time Management – Outputs may persist longer if deemed more relevant, or rotate out smoothly during high-frequency tasks
- The optimization layer and output slots can be manually toggled on or off by the user at any time, allowing for flexible screen usage and focus control. This ensures that advanced coordination features remain accessible without occupying screen space unnecessarily.

Core UX Interactions

Each display slot may include built-in controls:

- Freeze – Locks an output to prevent it from being overwritten
- Expand – Reveals more details, context, or reasoning
- Tree-of-Thought View – Displays alternative reasoning paths or similar insights the system considered but filtered

Human-in-the-Loop Controls

Most importantly, the system is designed to keep the human fully in control. At any time, the user may enter simple natural-language instructions such as:

"Keep this result visible for longer"

"Add a button to export as image"

"Refresh every 10 seconds"

"Hide all diagrams for now"

These textual overrides are processed by the Output Coordinator as part of its reasoning layer and are treated as live interface instructions, allowing seamless interaction without technical configuration.

This ensures that the UI layer remains not only intelligent and adaptive—but transparent, tunable, and collaborative.

Practical Example: Electrical Installation Business

A professional user—such as an electrician—can define a local folder as a live data source for one or more AI agents. This folder might contain:

- Wiring plans and layout diagrams

- Safety manuals and regulatory checklists
- Contracts, work orders, and customer correspondence
- Historical project notes or troubleshooting records

All files are indexed into a local vector database, allowing agents to retrieve relevant context instantly during reasoning.

When the user initiates a new task—such as drafting a project offer or reviewing a compliance-critical job—the system activates a coordinated AI workflow. Depending on the request, the agents may:

- Extract applicable safety or legal requirements
- Compare the task to previous projects to identify missing steps or recurring issues
- Suggest optimized layouts or resource allocations, based on learned best practices

Tasks can be flexibly distributed across local and external AI agents, with optional masking/demasking, data confiscation, and privacy filters applied where needed—allowing a secure and modular setup tailored to the user's preferences and risk profile.

This hybrid model balances performance, privacy, and adaptability. Simple lookups and compliance checks can run locally, while deeper reasoning or advanced planning can leverage external models—without compromising control or context awareness.

Local Execution & Privacy-First Design

The entire agent system can optionally be run inside a secure, sandboxed desktop app, giving users full control over:

- API-based cloud LLMs (e.g., OpenAI, Mistral, Claude)
- Self-hosted local LLMs (for partial or full offline operation)
- Private vector storage and memory indexing
- Full GDPR compliance and local data sovereignty

Summary

The **WR CODE** architecture forms a dynamic AI optimization layer that adapts to task complexity in real time. It empowers professional users—entrepreneurs, engineers, educators—to work with increased foresight, efficiency, and strategic depth.

Instead of waiting for users to ask the “right” questions, the system actively helps surface smarter paths—offering not just answers, but direction.

Additional Transparency and User Control via Local Configurable Interface

To ensure real-time oversight of the AI orchestration process, the system includes an always-active control panel that can be rendered in three flexible ways: as a collapsible panel within the browser extension, as a standalone full-page interface, or as a dynamic display slot. Acting as the system's 'brain', it continuously visualizes the live reasoning and decision paths of key agents such as the Input Coordinator, Output Coordinator, and Supervising Agent. All three display variants support customizable templates, allowing users to adapt the interface to their specific workflows and preferences. The panel runs entirely on-premise as part of the self-hosted orchestration environment and is fully configurable for advanced use cases.

Live Reasoning Inspection and Adjustment

The interface displays and allows real-time interaction with key system logic, including:

- Detected goals and subgoals, derived from user intent
 - Can be manually edited, toggled on/off, or locked to preserve critical objectives during reasoning
- Generated follow-up questions and reasoning paths
 - Each can be reviewed, edited, regenerated, or deleted individually
- Adjustable control fields, such as:
- Maximum number of follow-up questions
- Trigger conditions (e.g., on input change, rule match, or time interval)
- Freeform context prompts (e.g., "Do you have more background info to support this goal?")

- Enable/disable checkboxes for logic modules, output types, or agent roles
- Display port toggle for routing specific outputs to predefined display slots (e.g., main area, sidebar, external screen)

Output Control

The behavior of the Output Coordinator Agent—including display slot logic, refresh behavior, and user interaction settings—can be adjusted here as well. Users may also submit natural-language overrides (e.g., “Freeze this result,” or “Hide diagrams”), which are interpreted live by the system.

Optimization Utilities and State Awareness

- Quick Optimize buttons can be used to trigger strategic prompt improvements or apply higher-order transformations across the agent graph
- The system automatically detects new goals or context shifts, e.g., when starting a new task or input thread
- Transitions are surfaced transparently, with the option to manually reset or revert the current reasoning context at any time

Most of the logic displayed in the control interface is live-bound to the underlying DOM, meaning the change is reflected in real time within the running agent system. Rather than acting as a passive viewer, the user becomes an active manager—guiding the optimizer’s evolving thought process with precision and flexibility.

This control layer is essential for high-responsibility use cases where transparency, explainability, and fine-tuned AI alignment are non-negotiable.

Context Extension Prompts (with AI-guided gap detection)

To better understand the user's intent, the system includes AI-driven context extension prompts. These are not static fields — they are generated or adapted based on what the system identifies as missing or ambiguous information.

For example:

- If the detected user intent is a request to optimize a workflow but doesn't specify the tools used, the system might ask:
"Which platforms or apps are involved in this workflow (e.g., Notion, Outlook, Trello)?"
- Or if a goal is stated without timeframe or constraints, it might suggest:
"Do you have any deadlines, technical constraints, or preferred automation tools I should consider?"

These prompts are designed to:

- Actively identify potential context gaps
- Suggest relevant fields or clarification questions tailored to the task

- Help the AI refine its understanding of both short-term goals and long-term strategic intent

The system continuously makes its reasoning process transparent, empowering the user with full control. If the output is not as expected, the user can inspect and adjust the underlying logic in real time. All automatically detected context gaps, follow-up questions, and reasoning steps can be toggled on or off, edited, or overridden at any time—ensuring the system remains aligned with the user's intent.

Customizable, LLM-Driven Web Interface

The web-based interface is not static—it is **LLM-driven and fully customizable**. That means:

- The system is designed to support dynamic adaptation of forms, logic triggers, and UI components through configurable profiles tailored to specific domains or workflows (e.g., electricians, lawyers, engineers). Users will be able to create and switch between these profiles to activate relevant reasoning strategies and interface behaviors. In addition to personal configurations, the concept includes a curated library of **vetted community templates** created by domain experts. For more open-ended or cross-domain scenarios, the system will also offer **broad, flexible optimization templates** that apply general reasoning strategies without being tied to a specific field.
- The underlying logic (e.g., how subgoals are derived or how meta-questions are generated) is **controlled by LLMs**, which enables **semantic flexibility** far beyond static rule-based systems.

- Developers or domain experts can **extend the configuration templates** by integrating more complex reasoning steps, business rules, or specialized input fields. The long-term goal is to provide the user maximum flexibility in orchestration setup and UI control..
- These **custom interface configurations** (form templates, goal models, control panels) can be **shared with the community** or bundled as open components.

This opens the door to a **plugin-like ecosystem** where:

- Anyone can create and publish **domain-specific optimizers**.
- Communities can evolve best-practice reasoning models **collaboratively**.
- On-premise deployments benefit from a growing **library of modular UI + logic bundles**—without compromising privacy or vendor lock-in.

Spatial Distribution & Extended Input Channels

While fully functional on standard laptops, the orchestration system is designed for **workstations with multiple screens** and **VR devices equipped with built-in browsers**, where the extended display space enables clear spatial separation of agent outputs and uninterrupted parallel interaction.

Outputs can also be routed to **external endpoints** like AR glasses or mobile dashboards, allowing findings from one user's session (e.g., a backend analyst) to be delivered in real time to another (e.g., a field technician).

To support more complex or resource-constrained environments, the system allows multiple **master input tabs** to operate in parallel—enabling input from different sources, users, or subsystems without interrupting coordination.

In addition to standard input methods, the **WR CODE** architecture supports **sensor-level and system-state input**, including:

- Application state snapshots
- DOM structures from web interfaces
- Clipboard, pointer, and voice input
- Sensor feeds from AR/VR devices or robotics platforms

This extensibility allows the system to bridge physical and digital contexts in real time—making it suitable not just for analysis, but also for **situational coordination and cross-device collaboration**.

Example Use Case: From Simple Query to Informed Decision through Multi-Agent Orchestration

In this use case, a user interacts with the system via a chatbot interface and asks a seemingly simple question: "Can you find me a good USB stick?"

Rather than returning a single product recommendation, the system activates a parallel multi-AI agent workflow that expands the request across multiple dimensions—revealing smarter options, identifying hidden risks, and suggesting long-term strategies the user didn't explicitly ask for.

The process starts with the Input Coordinator Agent, which interprets the prompt, consults memory, and determines that deeper inquiry is warranted. It generates a set of optimized follow-up questions that break down the request into technical, contextual, and strategic subcomponents.

Each question is then routed to a specialized AI helper agent:

- One agent builds a manufacturer comparison table, evaluating flash type (SLC, TLC), controller quality, encryption, shock resistance, and warranty
- Another matches USB stick types to different user profiles (amateur, professional, enterprise), factoring in durability, write endurance, and portability
- A third explores alternative storage media such as M-DISCs, WORM optical formats, or enterprise-grade immutable storage—surfacing scenarios where long-term data retention is critical
- A visual agent renders diagrams comparing lifespan, usage patterns, and reliability under various environmental conditions

These results are reviewed by the Supervising Agent, which identifies redundant outputs, fills contextual gaps, and ranks relevance. The refined insights are passed to the Output Coordinator Agent, which dynamically arranges the content across screen-aligned display slots—prioritizing clarity and continuity.

As the user continues the session, the system adapts:

- New outputs replace outdated ones or fill available visual slots
- Relevant results remain longer on screen to support decision continuity
- Layout and transitions are managed smoothly to avoid visual clutter

Behind this, the agents rely on global memory—which may include user-provided data such as company size, data sensitivity, access requirements, or past decisions. From this, the coordinator can infer hidden intent structures. For example:

"The user's company handles internal legal documents and shares access across teams."

From this, the system might infer and act on secondary objectives such as:

- "Prioritize encrypted or access-controlled storage devices."
- "Recommend storage with long-term reliability under frequent use."

- “Suggest a hybrid approach that offsets the weaknesses of single-device solutions.”

In this case, the system might suggest:

“Pair a USB stick for day-to-day use with an archival-grade M-DISC stored in a fireproof case inside a bank locker—ensuring redundancy, physical security, and long-term resilience.”

These types of proactive, meta-level suggestions help the user avoid critical oversights—such as trusting vital data to a device that may fail after 10 years, when alternatives exist that last 100+ years.

Optimized Outcomes Through Collective Reasoning

What distinguishes this system is not just its ability to answer, but its ability to **reveal what the user didn't know to ask**. As a result, the user is no longer making a superficial purchasing decision, but a **strategic, informed choice**—avoiding the silent failure mode of relying on a storage medium that may degrade silently over time.

A Continuous Feedback Loop

Throughout the session, a **recursive feedback mechanism** ensures the system evolves with the user. If initial queries reveal gaps, contradictions, or missed opportunities, the supervising logic can generate additional follow-up questions, activate more agents, or reprioritize display slots. This **feedback loop ensures both question space and answer space are continuously refined**.

Final Outcome: Empowering Better Decisions

By the end of the session, the user has not just received product suggestions. They've been **guided through a structured, exploratory reasoning process**, supported by domain-relevant agents, coordinated for clarity, and presented in a form that helps them act with confidence.

In the specific scenario above, the system likely **prevented a flawed decision**: using a USB stick to store critical data long-term, unaware that such media can silently degrade over time. Without intervention, that decision could have resulted in **irreversible data loss**. Through orchestration, the user was nudged toward **more durable and appropriate alternatives**—even though they had no prior awareness of their existence.

This exemplifies the system's core promise:

Turning vague user queries into structured, high-quality decisions—through intelligent, modular, and adaptive AI collaboration.

Integration with External Services via Connector Protocols (e.g., MCP)

The orchestration tool can connect to external services—like email, calendar, task management, or CRM platforms—via standardized **connector protocols**. One prominent example is **Anthropic's Modular Context Protocol (MCP)**, but similar connectors exist across ecosystems (e.g., OpenAI Tools,

custom APIs, function calling, or webhook bridges). These connectors act as **adapters**, enabling AI agents to interact reliably and securely with external systems.

 Note: MCP itself is not a memory framework or agent host—it's a **context-passing protocol** that allows structured interaction with LLMs. The orchestration tool is **connector-agnostic** and can work with whichever integration protocol your chosen AI model or platform supports.

Multi-Input Architecture with Specialized Roles

The orchestration logic allows **multiple master tabs** to operate in parallel; each linked to a different input modality or task role:

- **Master Tab 1** might connect to a **voice input stream**, used for natural-language commands like:
 - “Display the client follow-up draft in slot 5.”
 - “Show today’s calendar in slot 2.”
 - “Copy the notes from slot 4 into slot 1.”
- **Master Tab 2** may run a **text-based chatbot** for research, Q&A, or strategic reasoning—entirely separate from UI control commands.

This separation enables **fast, non-blocking interaction**, where operational UI actions and deeper reasoning tasks can proceed in parallel.

Local LLMs for Low-Reasoning UI Tasks

To minimize API usage and boost performance, **simple orchestration commands**—such as:

- “Show the content from slot 3 in slot 1”
- “Freeze slot 5”
- “Move calendar view to slot 2”

can be processed entirely **by a small, locally hosted LLM**. These lightweight models handle basic intent parsing and slot routing with minimal latency—keeping cloud LLMs free for heavier reasoning tasks. This also enhances privacy and offline usability.

MCP Integration and Context-Oriented Orchestration

The orchestration layer in this system offers powerful support for modular, high-speed, and context-aware automation workflows. A key enabler of this architecture is the integration of external systems through connector protocols like MCP (Modular Context Protocol)—used by Anthropic—and similar interfaces available for other LLM platforms, including ChatGPT's connectors or even custom REST APIs. These connectors allow agents within the orchestration framework to interact directly with

external services such as email platforms (Gmail, Outlook), calendar tools (Google Calendar, Microsoft 365), project management apps, or CRM systems. When integrated correctly, the orchestration tool becomes not just a reasoning assistant, but a real-time operational interface.

Low-Level Reasoning with Lightweight Local Models

Simple routing tasks like "show draft in display slot 1" or "mirror output from slot 3 to slot 2" can be delegated to lightweight, locally hosted LLMs. This reduces latency and ensures that basic orchestration logic remains offline, transparent, and fully under user control.

Secure Human-in-the-Loop Execution

The orchestration tool does not force autonomy by default. Whether tasks are executed automatically or shown as drafts depends entirely on the template and system configuration. Secure templates prioritize:

- Displaying all automation steps (e.g., email drafts, calendar entries) in a visual display slot first
- Requiring explicit user approval via voice or button before final submission

That said, advanced users can configure fully autonomous flows if permitted by the connected service (e.g., n8n, custom APIs, or specific connector permissions).

n8n as a Seamless Integration Layer

An ideal companion to the WR CODE orchestration system is **n8n**—a source-available, locally hostable automation engine. While not required for basic operation, it extends capabilities with deeper backend logic, third-party integrations, and multi-step workflows. In addition, the orchestrator ships with **lightweight source-available automation modules** on both **mobile** and **PC**, selected per device profile—for example **Node-RED** (Apache-2.0), **Huginn** (MIT), **Actionsflow** (MIT), **Automa** (AGPL-3.0), and on **Android Easer/Termux** (GPL-3.0) for on-device triggers and scripts. These cover default automation needs out of the box; teams can later run **n8n** as an external companion when they need richer workflow design or enterprise integrations.

n8n integrates cleanly into this modular WR CODE architecture by:

- Acting as the backend task executor triggered by orchestration logic
- Running predefined automation chains (e.g., send an email, query a CRM, fetch and process a document)
- Responding to display slot routing or contextual instructions from helper agents

Since n8n operates within the browser, it can be opened in its own tab and authenticated locally. Once logged in, orchestration agents (e.g., controlled via user speech, text, or intent) can trigger

logic either through DOM interaction or direct API/webhook calls. This creates a bridge between user-level prompts and complex, autonomous backend automation — all under full user control.

Real-World Example: Alias-Based Orchestration and Real-Time Voice Overrides

The orchestration system intends to allow users to trigger complex, multi-agent workflows using simple aliases — such as: "Activate workspace747"

These aliases are predefined in a backend automation layer (e.g. using n8n etc.) and can launch full stacks of coordinated actions across agents, reasoning units, UI displays, and logic controllers. Each element in the system — whether an agent, slot, coordinator, or setting — is assigned a unique identifier, generated from customizable HTML templates that define the structure of the orchestration stack

. These templates include:

- Global and immediate user intent
- Reasoning traces
- Follow-up questions
- UI display slot routing with settings
- Ai agent behaviors

The result is a fully modular interface where every component — from individual control elements and AI agents to entire automation workflows — can be addressed, activated, and routed dynamically. Users can configure the system so that a specific unique ID, alias, or spoken name acts as a trigger. Depending on the setup, this can activate a listening AI agent, reorganize output slots, or execute backend logic — prompting helper agents, coordination agents, or supervising agents to respond accordingly. Even full automation workspaces can be launched in real time via voice, text input, or programmable keys (e.g., Streamdeck). These programmable buttons can also be embedded directly into the user interface through customizable templates.

- “agent_23 – rephrase to sound more confident”
- “output_coordinator – expand slot_5 and highlight key terms”
- “reasoning_intent_1 – I search for the firmware not for the bug”
- “supervisor_22 – trigger a steeprocketsearch747(alias) why I can’t burn the MDISC”

Voice commands can act as a parallel master control layer, enabling overrides, adjustments, or clarifications even while automated flows are running. Since everything is driven by backend automation and HTML-defined identifiers, the entire stack remains fully configurable, extensible, and reusable — making the system highly adaptable for different users, workflows, and use cases. The WR CODE architecture supports parallel input streams with unified or separate backend automation,

where display slots—**independent of the active master tab**—can be dynamically managed by AI using simple logic patterns or manually assigned by users to specific tasks for flexible, context-aware layouts.

Multi-Device Voice Input Integration

Voice commands are intended to be captured through a lightweight voice interface embedded in multiple endpoints:

- A **mobile app** (smartphones/tablets) with push-to-talk or passive listening
- **AR headsets** or wearable devices equipped with microphone access
- A built-in module within the **orchestrator app itself**, allowing direct control via **headset** or **microphone** input while working in the browser

Intent-Triggered Automation: Passive AI Detection

In more advanced usage, no direct prompt is even necessary. The orchestration system can passively infer user intent from contextual activity. For example:

A user records a meeting through a master tab interface.

- The system detects this is a structured team session and triggers:
 - Real-time transcription (via Whisper)
 - Context extraction: goals, risks, unresolved questions
 - Summarization and optimization suggestions
 - Gap and opportunity detection
- These results are compiled into a structured WRStamped PDF overview.
- The system drafts an email containing the meeting summary, WRStamped PDF, and personalized follow-ups to each participant.
- The draft email is automatically displayed in an available display slot for review and confirmation.

This entire chain can be executed without cloud dependency — running locally on user-owned infrastructure or in a private n8n instance, preserving full data control and privacy.



WR CODE in Action: Real-Time AI Orchestration During M&A Legal Negotiations

Scenario

Inside a confidential M&A negotiation between legal teams from two corporations, each group is represented by multiple senior attorneys. The stakes are high: cross-border compliance, intellectual property transfer, and post-merger risk allocation are on the table. Discussions are tense, technical—and every phrase counts.

Invisible AI Support

While the conversation unfolds naturally, WR CODE operates silently in the background. A local operator runs a **browser-based orchestration console**. No external cloud. No phone calls. No lag.

Behind the scenes, WR CODE coordinates a **multi-agent AI system**, each agent specializing in a domain:

- **Legal Risk Agent**: Monitors for exposure to undefined liabilities, anti-trust triggers, or unenforceable clauses
- **Regulatory Agent**: Flags cross-jurisdictional compliance concerns in real time (e.g., GDPR, FCPA)
- **Behavioral Agent**: Highlights patterns in phrasing and tone suggesting hesitation, evasiveness, or concessions
- **Comparative Agent**: Cross-references live discussion with prior drafts or negotiation history

AR Feedback Loop

Selected participants (e.g. lead negotiators) wear AR glasses discreetly connected to the **WR CODE** node. They receive **real-time visual overlays** cycling through:

- Top 3 flagged risks or inconsistencies
- Summarized intent shifts in the other party's language
- Recommended counterpoints or clarification prompts

No speech leaves the room. No trust is assumed. All agents run **air-gapped or containerized** on-prem infrastructure, ensuring complete control over inference and context.

Outcome

With WR CODE, negotiators gain **augmented legal foresight** —without interrupting flow or alerting the opposition. Silent, private, proactive. The orchestration turns passive hearing into active strategic advantage.

Summary: Dynamic Human-AI Collaboration

The **WR CODE** architecture redefines AI integration by enabling real-time, context-driven orchestration across modular agents — all triggered through speech, text, or detected behavior. By

leveraging n8n's automation layer in combination with WR CODE's orchestration logic, users gain an interactive, agent-driven environment where even highly complex workflows become intuitive and auditable.

Instead of isolated chatbots or opaque automations, the user sees exactly what happens — who does what, when, and why — and remains in control at every step.

Adaptive Orchestration: Mode-Based AI Stack Selection

To enable real-time adaptation across heterogeneous AI environments, we propose a codename-based orchestration layer. This mechanism allows users and systems to declaratively select execution profiles using short, semantic identifiers (e.g., codename cheap, codename deep, codename secure). Each codename corresponds to a predefined orchestration policy that governs which AI tools, models, memory modules, and services are permitted at runtime.

In standard deployments, the setup and configuration of the orchestrator is handled via formalized **automation templates**. These templates define toolchain composition, resource constraints, memory behavior, trust policies, and fallback conditions. While highly flexible, template-based orchestration requires a degree of technical expertise and manual preconfiguration.

The codename-based mode introduces an advanced layer designed to **lower the barrier to entry**. Instead of requiring full templates, users can invoke orchestrated tasks using minimal semantic input. The orchestrator interprets the selected codename and autonomously resolves:

- the appropriate AI tools or services,
- execution context and trust boundaries,
- optional memory integration,
- fallback and optimization strategies.

The orchestrator supports local, cloud-based, and hybrid AI service setups, including large language models (LLMs), OCR modules, code validators, document processors, visual or audio analyzers, and other specialized AI tools. All tools must be explicitly whitelisted to ensure secure and predictable execution. Each whitelist entry includes metadata such as tool identity, origin, runtime hash, and policy scope.

⟳ To address a broader range of operational scenarios, the orchestrator includes a **self-improvement mechanism**. It continuously monitors task performance, error patterns, and usage trends to refine orchestration behavior. Additionally, the system may optionally suggest *missing tools or modules* based on task context and prior user behavior. If enabled, these suggestions can be

retrieved from the internet or trusted repositories, depending on the user's **configured privacy level** (e.g., offline-only, internal catalog, internet-enabled discovery).

Users may further adapt orchestration behavior by explicitly training personal or organizational preferences, including:

- preferred AI tools or providers,
- obfuscation and privacy enforcement levels,
- optimization depth (e.g., single pass vs. multi-agent chains),
- allowed concurrency (number of simultaneous subprocesses),
- maximum execution time or fallback criteria.

These preferences can be stored locally, encrypted in a WRVault, and bound to a WRCode identity to ensure reproducible behavior across contexts. The active mode is visualized in the user interface and can be locked by administrators. To prevent accidental changes to critical settings, unlocking a mode requires a valid WRCode scan. Mode changes require administrative permission and may optionally be anchored on IOTA or similar distributed ledgers for tamper-proof auditability.

 From a technical standpoint, the orchestration logic can initially be implemented using a general-purpose large language model (LLM) to interpret user codenames, resolve toolchains, and generate

execution templates. This approach ensures transparency and flexibility without requiring upfront training. Over time, lightweight machine learning models may be introduced to optimize runtime behavior, predict tool performance, or rank fallback options. A hybrid strategy—combining LLM-based reasoning with structured ML-based optimization—offers the best balance between adaptability and explainability.

Importantly, this adaptive behavior extends beyond the orchestration layer itself. The entire chain of logic—including reasoning path selection, output coordination, display timing, and optimization strategy—is dynamically aligned with the user's intent and context. The user remains actively in the loop and can directly train the orchestration layer by adjusting preferences or providing feedback over time. This empowers both technical and non-technical users to shape the system's behavior according to evolving needs and priorities. While small local LLMs are generally sufficient for policy interpretation and orchestration tasks, more complex scenarios may benefit from a hybrid approach, where deeper reasoning and agent coordination are delegated to cloud-based or containerized models as needed.

By combining dynamic service patching with adaptive preference modeling and user-defined trust boundaries, this orchestration model balances security, transparency, and usability—supporting both casual users and regulated enterprise environments.

WR CODE & QR: QRGiraffe → From Static Codes to Orchestrated Intelligence

QR Codes are everywhere—but their default behavior is outdated and risky. Most systems execute QR payloads instantly: opening links, launching apps, or triggering silent actions. No inspection, no consent, no explanation.

QRGiraffe redefines this paradigm.

Instead of treating QR Codes as blind triggers, it transforms them into structured, intelligent entry points—processed securely, explained visually, and orchestrated through modular automation logic.

What Happens When a QR Code Is Scanned in **QRGiraffe**?

Local Capture, No Execution

- Upon scanning, the QR Code is parsed, encrypted, and stored locally within the **WR CODE**-enabled app.
- → No automatic execution of the payload takes place.
- → No outbound network request is triggered by default.

Explicit User Whitelisting

For a QR payload to proceed, it must be explicitly approved by the user.

There is no automatic trust decision.

Optionally, the **WR CODE** community can highlight that a source is known or cryptographically signed, but approval is always manual by the user itself. (Explicit Whitelisting)

Encrypted Orchestration Trigger

Once the orchestrator is online, the payload is sent securely for analysis.

The orchestrator activates a predefined workflow and the general optimization layer—completely local and privacy-respecting.

Visual Analysis & Contextual Insight

Unlike traditional QR tools, WR CODE uses the entire screen real estate to deliver a rich user experience:

- Full-page preview of destination intent
- Visual overlays explaining actions, risks, comparisons, suggestions, gap detection
- Structured metadata view
- Step-by-step automation reasoning
- Separate panels for content, trust signals, and suggested actions

The QR result is no longer a black box—it's an interactive, explainable environment, fully under user control.

Form Handling Policy: Preview Only

WR CODE never auto-submits forms. If a WR payload includes a form or the optimization layer pre-

fills a form for convenience, it is parsed and displayed in preview mode. Actions like submission, authentication are only possible after explicit user confirmation.

✉ Enhanced QR Experiences for Trusted Providers

WR Code providers can include structured instructions to display a visually organized overviews within the orchestration interface and provide additional data. (Workflow Ready Codes)

-  Digital signatures to prove authenticity
-  Structured metadata (validity, purpose, automation flags)
-  Semantic intent layers (Simple text instructions for AI-Agent automation), such as:
 - "Show additional video" (Default AI automation handles such tasks on premise)
 - "Draft an Email with this content and book a meeting in calendar as draft"
 - Open a signing interface (e.g., DocuSign) → Perfect for events, delivery handoffs, or mobile B2B transactions
 - "Display a visually organized discount overview"
 - "Show a 3D model in a separate slot"

Custom automation hooks for safe orchestration triggers

These enhancements help WR CODE present a smarter, richer user experience—without compromising trust or security.

Security Architecture: Groundbreaking by Design

QRGiraffe introduces multiple protective layers by default:

-  No execution without consent
-  User-controlled whitelisting
-  Transparent orchestration flows
-  Modular reasoning for every action

For advanced users, optional isolation layers are available:

The orchestrator can be run inside a dedicated virtual machine, booted from an external SSD, using a hardened operating system. This provides full execution separation from the host, useful for **high-risk environments or forensic workflows**. RAM-only operating systems offer extreme execution isolation by eliminating all disk-level traces, even within virtualized environments.

Designed for Full-Screen Orchestration

Unlike mobile QR scanners or browser redirects, the **QRGiraffe** concept uses every pixel to explain, visualize, and coordinate WR-based workflows.

-  Reasoning panels alongside content previews
-  Interactive controls for reviewing each step
-  DOM overlays, metadata inspection, automation logs
-  Risk and benefit indicators and origin validation
-  Adaptive layout for multi-monitor or widescreen displays

The result: a truly intelligent WR interface—where the scan is just the beginning of a controlled, explainable process. (WR Codes are Workflow Ready Codes based on the QR Code technology)

Integration with Distributed Ledger Technologies (e.g., IOTA, Solana, Optimism)

QRGiraffe includes a conceptual mechanism for secure, decentralized logging and verification through cryptographic anchoring on distributed ledger networks such as Solana, Optimism, or IOTA. The choice of

anchoring strategy is dynamically adapted to the specific use case, considering factors such as, latency, cost, and required auditability.

- Payloads such as public URLs, structured metadata, or AI automation instructions can be selectively published or verified via high-performance blockchains such as Solana, depending on the use case and required latency.
- Only publicly shareable, non-personal information is eligible—no personal data, user identifiers, or location-based metadata are stored.
- Anonymity is preserved by design, with no technical linkage to individuals, organizations, or geographic locations unless explicitly configured.
- Users remain in control and may activate this feature as needed by supplying their own API keys, node URIs, or access credentials.

 **Example: Claiming a Discount via WR Code with Offline Execution and Secure Delayed Synchronization**

- A user scans a WR Code printed on a product flyer to activate a limited-time discount. The WR Code can be scanned from any supported device — mobile phone, desktop browser, or a hybrid orchestrator environment.

Instead of immediately redirecting to a website or submitting personal data, the WR Code follows a privacy-preserving and verifiable execution model:

1. Local Parsing and Execution (Offline-Capable)

- The WR Code payload (e.g., offerID=SUMMER10) is parsed and a claim fingerprint is generated locally:
SHA256(offerID + 2025-07-02T12:03:00Z)
- This fingerprint is securely stored on the user's device — e.g. in browser storage, local WRVault, or native app memory — without transmitting any data.
- If the template was previously cached or embedded, the WRCode automation may already run locally (depending on the device's trust policy).

2. Cryptographic Validation After Reconnection

- When an internet connection becomes available, the orchestrator securely:
 - Downloads the required WRCode template and template map, context and eventually required code snippets, including policy, versioning, and WRStamp metadata.
 - Verifies all components cryptographically, using WRStamp signatures anchored to a public ledger to ensure authenticity and prevent tampering.

- Resumes or completes the workflow (e.g., issuing the discount code), based on the verified claim and stored context.

3. Tamper-Proof Anchoring on IOTA (Optional)

- As part of the claim flow, the hash of the scanned WR Code + timestamp may be anchored to the IOTA Tangle.
- This operation occurs asynchronously and does not delay the user experience.
- It serves as a public, immutable audit log proving that a claim attempt occurred — without revealing the user's identity or device data.

4. Cross-Device, Secure-by-Design Architecture

- WR Codes function identically across mobile, desktop, and orchestrator-integrated systems.
- No automation template is ever executed unless:
 - It is signed with a valid WRStamp
 - It matches the policy defined by the orchestrator or trust layer
 - It is verified at runtime using cryptographic checks

 The user stays anonymous, and the offer can be verified without tracking or profiling.

 Example: Access via WR Code Invitation (Co-Working or Conference Use Case)

A visitor receives a WR Code-based invitation to enter a co-working space or event venue.

Upon scanning the WR Code:

1. Local Parsing and Fingerprint Generation

- The WR Code contains a signed payload (e.g. invitationID=EVT-7421) and a WRStamp.
- The device computes a deterministic fingerprint using a secure timestamp:
SHA256(invitationID + timestamp)
- This fingerprint is either stored locally or processed transiently, depending on device policy.

2. Trust Evaluation and Policy Enforcement

- The WRStamp is cryptographically validated against a trusted publisher key or verified via a connected orchestrator.
- The access policy is evaluated:
 - Is the invitation valid for the current time window?

- Does the signature match a known publisher?
- Is the associated role or token included in the locally cached or distributed whitelist?

3. Access Grant and Optional Anchoring

- If the validation passes, the access system unlocks the gate or issues a one-time access token.
 - Optionally, the claim hash is anchored to a distributed ledger (e.g. IOTA) for tamper-proof auditability — without disclosing user data or device identifiers.
-

Summary:

WR Codes enable secure, verifiable access control without requiring constant online validation.

All logic is driven by signed templates and policy-bound instructions, verified at the edge or through the orchestrator.

The cryptographic method and anchoring strategy are selected per use case — ensuring performance where needed and provability where required.

 No personal information or credentials are transmitted—only a cryptographically verifiable claim is processed.

 Example 3 – WRCollect: Privacy-Preserving Loyalty Points

A customer receives a WRCollect code on a printed or digital receipt, containing signed loyalty point data.

- The code is scanned and hashed locally, then stored on the user's device (e.g. browser or mobile vault).
 - On future visits, the stored hash is re-presented for point redemption and verified against the original WRStamp or a trusted policy.
 - Optionally, the hash is anchored to IOTA for tamper-proof proof of claim — without exposing personal information.
-  **No login or registration required — WRCollect enables anonymous, verifiable loyalty tied to signed templates, not user accounts.**

Example 4 – WRConnect: Secure Device Pairing

A user scans a WRConnect code from a new IoT device (e.g. router, sensor) to initiate onboarding.

- The pairing request is processed locally by WR CODE without exposing setup data.
- The WRCode payload is verified cryptographically using a signed template — no external lookup required.
- Optionally, a hash of the pairing event is stored or anchored (e.g. to IOTA) for later auditing.

 **Devices are added securely and privately, with no sensitive configuration data sent to third parties.**

Forward-Looking Vision: Enabling Zero-Knowledge Verification

Although not yet implemented, the architecture is designed to support zero-knowledge proofs (ZKPs) in future versions.

In this model, the system could:

- Coordinate local proof generation (e.g., "I am over 18", "I hold a valid credential")
- Accept and verify ZK proofs without revealing any underlying data
- Trigger logic based on verifiable claims — without ever seeing who the user is
 - ✓ Even in these advanced use cases, the user's identity is never exposed to WR CODE or any of its services.
 - ✓ All sensitive operations occur locally and client-side, maintaining strict privacy by design.

Extendable to Other Domains Beyond QR

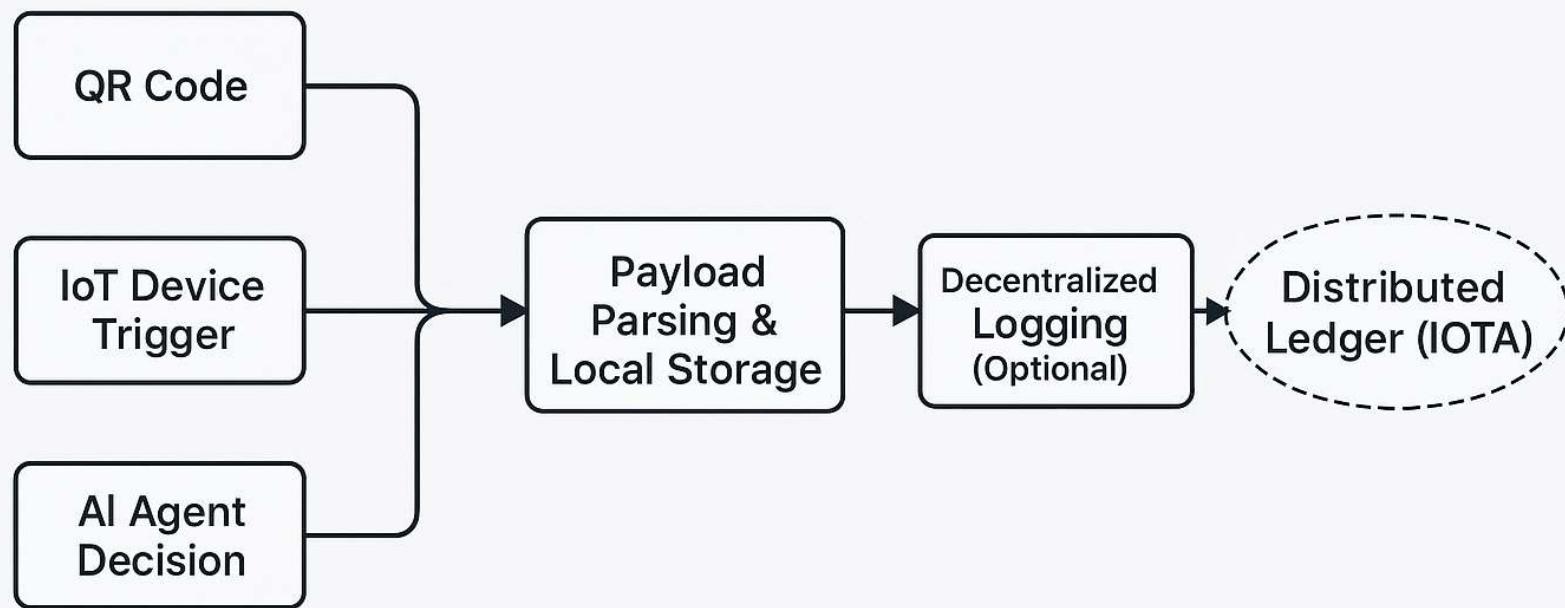
This decentralized verification approach is not limited to QR interactions.

→ It can also be applied to other components of the **WR CODE** system, such as:

- IoT device triggers or smart home automation routines
- AI agent decisions and reasoning chains
- Audit trails for orchestrated workflows or inter-agent communication

→ This enables cross-domain consistency, verifiability, and long-term integrity of automation processes in dynamic and distributed environments.

OPTIONAL DECENTRALIZED LOGGING BEYOND QR INTERACTIONS



To enhance security, integrity, and privacy in AI-driven automations, the **WR CODE** framework introduces a local-first execution model where every QR Code is treated as potentially unsafe by default. Each scanned **WR Code** is displayed to the user in readable form before any action is taken, ensuring that no AI-triggered automation is executed without explicit user confirmation.

Users can make informed decisions when scanning QR Codes from well-known providers, even if those codes are not explicitly whitelisted by the community. Public whitelists and individual trust decisions can coexist, giving users full flexibility and control over execution.

To further strengthen security, **WR CODE**'s orchestration templates include automated checks for phishing risks and other potential threats before executing any **WR Code** actions. These security templates are fully configurable using simple, human-readable language, enabling both technical and non-technical users to define custom safety rules for AI automations.



Verified Template Installation, Personalization & Execution (WRCode Standard)

Every WRCode refers to a **unique Project ID**, which represents a discrete automation bundle registered on wrcode.org. Each project groups one or more **AI templates**, and each individual template within the project is assigned its own **unique TemplateID**. This structure allows WRCode automations to scale from single-use flows to orchestrated multi-step processes across devices—while remaining **fully auditable, tamper-proof, and trustless by design**.

All templates associated with a WRCode project must be:

- Published under a valid Project ID
- Individually registered and hashed on wrcode.org
- Permanently visible and cryptographically anchored
- Immutable unless explicitly versioned and republished

Only templates and embedded context that meet these criteria will function in WRCode scanners. If a template is missing, altered, or unpublished, it will simply not execute. This enforcement ensures that every automation executed through a WRCode is **100% transparent and verifiably safe**—not just for developers, but for end users and auditors.

When a WRCode is scanned, the corresponding Project ID is used to resolve all linked templates. These are downloaded **directly and exclusively from wrcode.org**, and parsed in a **sandboxed RAM-only runtime environment**. Templates contain no code—only structured JSON with orchestration instructions and pointers. They are parsed, not executed, ensuring deterministic, auditable behavior with **no privileged backdoors or hidden logic**.

By default, users may **retain downloaded templates** for transparency or future reuse. However, if users **modify templates** and want to reuse them in future WRCode sessions, they must:

1. Assign a new local Project ID on their device
2. Publish the new Project ID and associated TemplateIDs to wrcode.org

Templates and context will not function through the WRCode system unless they are published under a registered Project ID and meet full verification criteria. This includes both original and user-edited templates. The PoE mechanism proves that only exact, hash-matching templates can be installed and executed in a verified session.

Advanced users may choose to manually add further templates into the **orchestrator's optimization layer**, outside the WRCode scanner. This is fully permitted—but occurs **at the user's own risk and responsibility**. These additional templates:

- Will not be triggered by WRCode scans
- Will not benefit from PoE-based verification
- Will not be governed by WRCode integrity policies

Instead, they function like local, user-managed agents that complement the WRCode-driven flow. This allows for deep customization within orchestrator environments (e.g., WR CODE), without compromising the **trustless, transparent execution model of WRCode templates**.

Critically, only **Pro users** in the WRCode system are allowed to:

- Edit templates and republish (local modification is always possible)
- Register new Project IDs
- Publish or share automation bundles with the community

This restriction ensures that only accountable, traceable participants can expand the system, maintaining a **balance between openness and safety**. All published templates remain publicly inspectable and cannot be removed or silently modified once released.

In this architecture, WRCode evolves into a **secure gateway to user-driven AI automation**—combining decentralized trust, cryptographic validation, and fine-grained personalization under a unified, auditable standard.

Note: Templates and Project IDs not published via wrcode.org **will not work** in WRCode scanners. Private flows must use manual orchestration paths and cannot leverage the public PoE system.

This approach is particularly valuable in the context of AI-triggered automations. Unlike traditional WR Code uses—where malicious codes typically result in simple redirection or phishing—AI-driven automations carry higher risks. A spoofed **WR Code** could trigger unauthorized actions or malicious workflows without user awareness. It is important to highlight that WR Codes are already exploited in the real world for malicious purposes, including phishing attacks and malware infections. Cases have been reported where altered **WR Codes** on shared vehicles or public terminals were used to deploy malware or steal sensitive information.

Summary: Trust, Clarity, Control

With **QRGiraffe**, QR Codes are no longer a risk—they're an opportunity to activate intelligent, modular workflows with:

- Full semantic understanding
- Deeply layered protection
- Visual orchestration across the multi-screen setups
- Optional trusted provider integrations
- Future-ready ZKPs, adaptive architecture

QRGiraffe transforms static QR Codes into dynamic workflow-ready, orchestrated user experiences—intelligent by design, controlled by you.

Per-Customer AI Agents in WRCode: Local-Only Orchestration with Embedded Context and Memory

In the WRCode ecosystem, automation flows are orchestrated using **template-maps and publisher context**—signed, hash-verifiable files that describe the complete set of execution paths for a given workflow. Instead of requesting a tailored map from a remote service based on device preferences or capabilities, WRCode follows a **local-only resolution model**:

- The full, signed template-map is downloaded once (or loaded from cache if previously retrieved).
- The orchestrator evaluates all available paths locally, selecting the most appropriate composition based on the user's configured preferences, available LLMs, and network policy.

- No preferences, capabilities, or telemetry are transmitted upstream.

This approach ensures **zero data leakage** during orchestration setup, maintains full offline capability, and keeps all decision logic under the user's direct control.

Per-Customer, Real-Time AI Agent

WR CODE instantiates a real-time, per-customer AI agent for each session. The agent is composed locally by the orchestrator from WRCode templates, public, verifiable context embedded on WRCode.org, and the user's private, on-prem embeddings (PC, smartphone, or other nodes). It executes under the local orchestrator and renders outputs into configurable display grids within helper tabs; agents themselves do not run in tabs.

Why this model

- Privacy & State: Session/long-term memory stays local; no sensitive data is pushed to WRCode.org or shared across tenants.
- Deterministic Context: Public manuals/SOPs are auditable & versioned on WRCode.org; internal documents are embedded locally. At run time, the orchestrator composes the active context and retrieves it (API for public indexes; RAG when using a local LLM).

- Adaptation without Drift: Behavior evolves via local prompt/tool/routing overrides. Local edits are marked as modified and are never auto-overwritten by new scans without explicit user approval.
- Compliance & Residency: Meets “approved-dataset-only” workflows: provenance is provable (WRCode.org version IDs + local doc IDs), data residency is maintained on-prem, and BYOK is supported for API credentials.

This real-time, per-customer assembly preserves privacy and compliance, avoids multi-tenant state bleed, and keeps operations transparent and auditable—without sacrificing the flexibility users expect from an “agent.”

In this model, the **publisher** still defines the core workflow and public knowledge base, but the **user** augments it with private intelligence, resulting in a **hybrid reasoning model**:

- **Public Layer:** Hash-verified context bundles (manuals, SOPs, installation guides) published openly for transparency and auditability.
 - **Private Layer:** User-owned memory and private context, stored in the local WRVault and embedded into the LLM during runtime.
-

Template-Map with All Paths Included

A WRCode template-map in this model contains **every valid orchestration path**, regardless of whether it will be used in the current environment. Each path specifies:

- Template IDs and role assignments
- Pointer graph (execution flow between templates)
- Required agent references (publisher agents, user agents, or both)
- Context bundle references (public, private, or merged)
- Display modes and prioritization for multi-output setups

Example (simplified):

```
{  
  "map_id": "tm-2025-08-15-001",  
  "compositions": [  
    {  
      "id": "local-default",  
      "path": "path_id_1",  
      "path_label": "Path 1",  
      "path_order": 1,  
      "path_type": "Normal",  
      "path_desc": "A normal path from node A to node B.",  
      "path_nodes": ["A", "B"],  
      "path_activities": [{"name": "Activity A", "x": 100, "y": 100}, {"name": "Activity B", "x": 200, "y": 100}],  
      "path_transitions": [{"label": "Transition 1", "x": 150, "y": 150, "x2": 200, "y2": 150}, {"label": "Transition 2", "x": 200, "y": 150, "x2": 250, "y2": 150}],  
      "path_start": "A",  
      "path_end": "B",  
      "path_is_start": true,  
      "path_is_end": false  
    },  
    {  
      "id": "local-default",  
      "path": "path_id_2",  
      "path_label": "Path 2",  
      "path_order": 2,  
      "path_type": "Normal",  
      "path_desc": "A normal path from node A to node C.",  
      "path_nodes": ["A", "C"],  
      "path_activities": [{"name": "Activity A", "x": 100, "y": 100}, {"name": "Activity C", "x": 300, "y": 100}],  
      "path_transitions": [{"label": "Transition 3", "x": 150, "y": 150, "x2": 200, "y2": 150}, {"label": "Transition 4", "x": 200, "y": 150, "x2": 250, "y2": 150}, {"label": "Transition 5", "x": 250, "y": 150, "x2": 300, "y2": 150}],  
      "path_start": "A",  
      "path_end": "C",  
      "path_is_start": true,  
      "path_is_end": false  
    }  
  ]  
}
```

```
"templates": [1257],  
"agent_refs": ["agent.user.printer.local.v1"],  
"context_refs": ["ctx.printer.manual.v3", "ctx.user.private.docs.v1"]  
},  
{  
    "id": "hybrid-extended",  
    "templates": [4278, 4547],  
    "agent_refs": ["agent.publisher.printer.v1", "agent.user.local.v1"],  
    "merge_policy": "stack_context_and_memory",  
    "context_refs": ["ctx.printer.manual.v3", "ctx.faq.v2", "ctx.user.private.docs.v1"]  
}  
,  
{"resolution": { "mode": "local_only" }  
}
```

Here, the orchestrator decides—entirely on-premise—whether to run a pure local agent, a hybrid chain, or any other valid composition.

Embedding Context into Configured LLMs

When a template-map path is selected, the orchestrator:

1. **Fetches public context bundles** referenced in the composition from the publisher's open repository.
2. **Retrieves private context bundles** from the local WRVault.
3. **Embeds both** into the configured LLM(s), which can be:
 - A fully local model
 - A cloud-hosted model
 - A hybrid of both

This embedding process enables the AI agent—whether local, cloud, or hybrid—to reason with **complete, domain-relevant knowledge** during orchestration. Because the private layer is never published or transmitted, the **privacy boundary** is maintained even in cloud-based setups.

Proof of Execution (PoE) and Verifiability

PoE extends to capture the complete **execution context** for every session:

- **Template-Map ID** and hash
- **Composition ID** selected locally
- **Agent Manifest hashes** (base-model reference, instruction set, runtime policy)
- **Public Context Bundle hashes** (openly published for audit)
- **Private Context Bundle hashes** (locally stored; verifiable by the user)

This structure ensures that anyone can:

- Verify exactly **which public context** was used
 - Confirm that **private context** was present without revealing its contents
 - Reproduce the orchestration path for compliance or dispute resolution
-

Security and Privacy by Design

- **No Server Hinting:** The server never receives user capability or preference data.
- **Open Public Context:** All publisher-provided context bundles are public, signed, and anchored for tamper-proof verification.
- **Controlled Private Context:** Private bundles are never transmitted; they remain in the WRVault and, by default, are only embedded into the LLM in the user's local environment. The user can explicitly choose to embed them in cloud-based AI agents, but always retains full control over when and how this occurs.
- **Deterministic Audit Trail:** Every execution can be reconstructed and verified using PoE.

In summary, the **per-customer AI agent** model in WRCode provides maximum flexibility and privacy: users get the benefit of public, auditable intelligence while retaining the ability to integrate sensitive, proprietary knowledge directly into their agents. Coupled with a full-path template-map and local-only resolution, this architecture delivers a **frictionless yet verifiable** automation experience that works equally well offline, in hybrid deployments, or with trusted cloud LLMs.

🔑 Geofence-Orchestrated Intelligence

Privacy-Preserving Location Triggers with Zero-Knowledge Verification and Intent-Aware Mobile/AR Automation

WR CODE introduces a new paradigm in geofence-driven automation: combining **zero-knowledge proofs (ZKPs)**, **on-device geolocation triggers**, and **intent-aware AI workflows**—without reliance on cloud infrastructure or surveillance-based tracking. Unlike traditional geofencing tools, which often link location to identity and upload data to centralized servers, WR CODE handles detection, verification, and orchestration either **locally or within a trusted private backend**, making it ideal for **wearables, AR/AI Glasses, and smartphones**.

🔒 Zero-Knowledge Entry Proofs – No Identity, No Coordinates, No Surveillance

As a user approaches a defined physical location (e.g., a logistics gate, event zone, or facility entrance), their device detects entry via GPS, Bluetooth, or Wi-Fi. Instead of revealing precise coordinates or user identity, WR CODE locally generates a **cryptographic hash**:

SHA256(zoneID + ISOtimestamp)

This functions as a **proof of presence**—optionally anchored on a ledger like Solana—but does **not disclose who the user is**, or where exactly they are. This makes geofencing usable for sensitive environments (e.g., factories, labs, field operations) while remaining fully privacy-preserving.

ZKP Integration Strategy

WR CODE supports modular integrations of established privacy-preserving techniques. Where required (e.g., under GDPR or AI Act compliance contexts), users may implement zero-knowledge proof flows using standard open source protocols such as zk-SNARKs (via libsnark or circom), Bulletproofs, or verifiable credentials via Hyperledger Aries (Idemix). These are not required for core system functionality and are left to the user's discretion based on use case and trust requirements.

WR CODE itself does not introduce novel cryptographic primitives. All optional ZKP integrations rely on public, licensed cryptographic frameworks and are compatible with privacy-first design principles.

Example: Privacy-Preserving Retail Offer with Anonymous ZKP Verification

A retail store offers an exclusive online discount that is only accessible to users who have physically visited the store (and maybe even scanned a WR Code additionally). The process is built to ensure complete privacy—without any GPS tracking or identity linkage.

How It Works:

1. When the user physically enters the store, their smartphone detects presence locally—via geofencing, GPS, or WR Code. (visit a local event or representation first)
2. The phone generates a Zero-Knowledge Proof (ZKP)—a cryptographic hash confirming that the event occurred.

3. This **hash is:

- Stored locally on the device (e.g., in the user's storage).
- Optionally anchored on a public blockchain (e.g., Solana) for verifiable, tamper-proof proof of presence.

No personal data is collected.

No GPS tracks, no timestamps, no identity—only an anonymous presence hash exists.

The store has no access to any data at this stage.

4. Later—when the user chooses to buy online—the retailer's system can:

- Ask the user to provide the presence proof (the stored hash).
- Compare it against the public Solana anchoring to verify eligibility.

5. If the hash matches, the user is whitelisted for the exclusive offer.

• The retailer only sees the valid proof—but has:

- ✗ No knowledge of who visited.
- ✗ No knowledge of when they visited.
- ✗ No way to track or correlate users to physical behavior.

If the user decides to purchase, they may naturally provide PII during checkout (name, email, shipping)—but this remains decoupled from the original physical visit.

 Key Privacy Advantages:

- No tracking: Geofence detection stays entirely local—nothing is transmitted when presence is detected.
- Anonymized proof only: The vendor sees only a hash—never location, time, or identity.
- User-controlled verification: Only the user can later choose to reveal a valid proof to access the online benefit.

In short:

- Physical presence is provable, but not traceable.
 - Vendors can verify “someone” was there—but never know who, when, or where exactly.
 - Users control both timing and identity disclosure separately.
-  WR CODE & WR Code Automation Sharing: Unlocking the Power of QR for Portable, Privacy-First Workflows

While QRGiraffe has already introduced a new era of secure and explainable QR interactions, an equally powerful capability is the ability for WR CODE to generate QR "Workflow-Ready Codes (WRCodes)" for seamless automation sharing. This approach enables entirely new forms of portable,

decentralized, and context-aware automation delivery—without reliance on centralized services or invasive tracking.

Importantly, WR Codes generated by **QRGiraffe** utilize standard QR Code technology (ISO/IEC 18004). This is not new technology in itself—the innovation lies in the unique way this existing technique is utilized to enable secure, explainable, and privacy-respecting automations. The term "**WRCode**" (Workflow-Ready Code) is used descriptively to highlight the workflow-readiness of the generated WR Codes. It does not represent a new code format or suggest any modification of the underlying WR Code standard—it is simply a conceptual layer on top of globally recognized QR technology.

In addition, WR Codes can be **dynamically generated** by digital services, software agents, or even robots—allowing real-time, system-agnostic deployment of automation workflows across diverse physical or virtual environments

From Static Codes to Dynamic, Shareable Automations

With QR WRCodes, users can:

- Export a **preconfigured WRCode** with embedded defaults and user-local personalization via **WRVault**. The QR remains a **signed pointer to a trusted automation**, not a runtime result. If the goal is to share automation *outputs*, they must be **re-signed and optionally anchored** as WRStamped snapshots. Include key initial inputs or settings within the QR
- Trigger automation that is fully personalized through local PII (personally identifiable information) already stored in the user's **WRVault**
- Enable the automation to explicitly request missing input from the user before execution

The result is a truly intelligent QR system that not only shares information, but shares action—empowering users to execute meaningful processes immediately, while retaining full control.

How It Works (Runtime WRCode Execution with Solana Validation)

1. Template-Linked WRCode

- The WR Code does not embed payloads directly. Instead, it contains a signed reference (e.g. ProjectID, version, publisher, SubID) pointing to a remote set of automation templates and context hosted on wrcode.org.
- Upon scan, the orchestrator fetches the templates and its manifest in real time and validates them against a Solana-anchored WRStamp.

- Supplemented by a vetted source-available code library, enabling modular on-premise extensions. This allows secure business logic to run locally — without exposing data — while still delivering a tailored, intelligent user experience.

2. WRVault-Based Personalization

- Inputs like name, department, language, or saved preferences are injected locally via the user's WRVault—fully offline and without exposing any personal data.

3. Dynamic Input Completion

- If required fields are missing, the orchestrator prompts the user in context (e.g. "Specify quantity" or "Choose reason") before allowing execution.

4. Verified, User-Controlled Execution

- Execution only proceeds once:
 - The template is verified (WRStamp + Solana anchor),
 - The policy conditions are met, and
 - The user explicitly approves the previewed action.
- This ensures full transparency and runtime control.

5. Optional Anonymous Logging & Proofs

- If enabled, the orchestrator generates a hash of the interaction (e.g. templateID + timestamp + anonymized metadata) and logs it:
 - On IOTA for long-term, tamper-proof audit trails
 - Using ZKPs for verifiable claims without exposing sensitive data
6. Optionally enables privacy-preserving location confirmation, proving that a device was within a predefined zone without exposing GPS data or traceable coordinates.

Key Advantages

WR Codes now act as trusted automation triggers, not data containers.

All templates are validated at runtime, anchored on Solana, and combined with local context from WRVault to enable secure, privacy-first workflows across any device.

WRCode as Secure App Trigger

WRCode is extremely flexible — beyond launching automations or downloading instruction templates, it can also act as a secure entry point for full applications.

When used in App Mode, a WRCode doesn't just trigger a static task — it establishes a verified session between the user and a WR-enabled application. This session can include:

- Login handshake, secured through a signed WRStamp
- Feedback loop, where AI agents exchange instructions and context in real time
- Template-driven orchestration, hosted on wrcode.org for trustless verification

Only pre-approved, tamper-proof templates can initiate such app-level workflows, ensuring that even complex use cases like dynamic AI apps remain auditable, privacy-respecting, and cryptographically verifiable.

High-Value Use Cases for WR Code Workflow Sharing:

| Use Case | Description | Value |
|---|---|---|
|  Field Inventory & Maintenance | Technician scans WR on equipment → automation pre-fills with stored user data + | Saves time, reduces errors, offline-capable |

 Legal Document Automation

prompts for piece count/status → generates instant report

 Coupon & Offer Claims

Marketing flyer with WR triggers personalized coupon claim → automation uses stored user data + asks for optional preferences → generates offer

 Access Control & Visitor Management

WR-based access request triggers automation that uses stored PII → asks visitor for reason/ETA → processes entry approval offline or hybrid

 Automotive & IoT Pairing

Scanning WR on device starts onboarding automation → local data pre-fills known fields → user adds any missing details → device is securely paired

Accelerates legal/admin processes while ensuring compliance

Drives engagement without forcing app installs or logins

Boosts security, reduces friction, maintains privacy

Simplifies device setup without cloud reliance

⌚ Geofenced Event Automation with Anonymous Logging

WR at physical location triggers time - or location-sensitive automation → event is logged anonymously on IOTA → optional Zero-Knowledge Proof ensures claim validity

Enables privacy-preserving location-based workflows and verifiable event logs

💡 Strategic Advantages:

- **Privacy-First by Design:** No personal data is embedded in the QR—only the automation logic or references.
- **Offline & Resilient:** Automation execution is possible even without an active internet connection but the initial verification always requires an active internet connection as part of the security concept.
- **Portable & Shareable:** WR Codes can be printed, shared digitally, or embedded in products — acting as lightweight triggers that reference publicly hosted, tamper-proof verified, auditable templates on platforms like wrcode.org, with no need for private or proprietary backends.
Time-Saving & Error-Reducing: Pre-filled forms, context-aware prompts, and automation minimize manual errors and speed up processes.

- **Verifiable Without Tracking** : Optional Solana and Zero-Knowledge Proof integration allows for cryptographically verifiable events without compromising user anonymity.

Real-World Impact:

By enabling the creation and execution of portable, personalized automations through QR, WR CODE empowers:

Business workflows that run with no infrastructure — or integrate seamlessly with existing systems.

Customer interactions that prioritize **privacy and ease-of-use** , without requiring accounts or logins.

Industry use cases where security, speed, and adaptability are critical — whether triggered via browser, standalone device, or a publisher's own **WRCode-compatible app** .

The QR becomes more than a link—it becomes an actionable gateway to orchestrated intelligence.

For organizations, event organizers, field services, and many others, this capability creates a new automation frontier: one where privacy, security, control, verifiability, and intelligence are not trade-offs, but standard features.

Security Concept for PII Protection and Automations in WR CODE (QR-Giraffe Module)

1. Objective

WR CODE follows a strict privacy-by-design approach to protect users from identity theft, IP theft, extortion, and data misuse. This security concept ensures that personal identifiable information (PII) and other confidential data remain fully under the user's control at all times.

2. Local Encrypted Vault

- PII and secrets are stored exclusively in a locally encrypted vault.
- Encryption used: AES-256-GCM or comparable secure methods.
- By default, highly sensitive PII (such as names, addresses, financial data) is never prefilled, displayed, or exposed during automated processes. The insertion of PII into any form occurs only at the exact moment of manual user confirmation, - To support cases where AI-generated entries or complex automations are involved, an additional option allows users to temporarily decrypt and preview specific non-PII content through a secure visual overlay layer prior to submission. This allows verification of dynamically generated data while maintaining strict separation from sensitive PII.
- Decryption and transmission of any data take place only on-the-fly and under explicit manual consent.

- To further strengthen control, the Vault supports adaptive authentication mechanisms based on security level:
 - For routine low-risk automations (Trust Levels 1 & 2): temporary automation access may be allowed without friction, provided no PII is involved.
 - For PII insertion, Vault modifications, identity resets, or any sensitive operation (Trust Level 3): biometric authentication (e.g., Face ID, fingerprint), passphrase, or hardware token is mandatory.
- Biometric authentication is particularly suited for frictionless yet secure experiences on smartphones, which represent the majority of WR Code use cases.
- Users without biometric capability may use a secure fallback such as a PIN code or hardware security key.
- The PII Vault is designed to evolve into a broader personal credential manager, capable of securely storing passwords and authentication tokens. This future capability will allow users to automate sign-up processes, logins, and password handling within orchestrated workflows. While not all features will be available from day one, the Vault is conceptualized as a central pillar for secure identity management and seamless automation. This capability is part of the long-term vision and may not be available in early versions of the system.

- To enhance granularity and privacy, the **WRVault** system is not a single container but a modular architecture composed of multiple isolated vault units—each designed for a specific category of user data. These include: a Critical **WRVault** for highly sensitive information such as personally identifiable information (PII), passwords, access tokens, and internal identifiers; a Sensitive **WRVault** for complex but non-critical documents such as medical reports or financial statements; and a Non-Sensitive **WRVault** for general-purpose data like user preferences, past interactions, or instructional context. When users upload documents—such as .txt, .json, or .md files—into the Sensitive or Non-Sensitive vaults, their contents are automatically embedded in the background into lightweight (~1 GB) local LLMs specific to that vault. This enables fast, on-device semantic retrieval and contextual reasoning. In contrast, the Critical **WRVault** is intentionally not equipped with any LLM and cannot embed or transform its contents. It is reserved exclusively for structured, non-embedded storage of critical fields like names, credentials, or customer numbers, which remain accessible only through direct user approval. Users must never upload PII or credentials into any vault other than the Critical **WRVault**. To reduce accidental exposure, the system includes automatic detection and masking mechanisms: if sensitive patterns (e.g., names, passport numbers, emails) are detected in uploaded files, the content is either rejected, masked or filtered before embedding. Users are notified of such events and can reassign the file to the correct vault. This design enforces strict separation between sensitive and contextual data while preserving

semantic capability where appropriate. Users remain fully in control of how data is classified, but critical fields are always handled under the system's strongest privacy protection.

- To further enhance security and future-proof the Vault against emerging threats such as quantum computing, the Vault can optionally be hardware-bound through a device-specific fingerprint combined with a cryptographic-based hash. This mechanism ensures that even if the encrypted Vault is exfiltrated, it remains unusable on any other device due to a mismatch between the stored hash and the hardware identifier.
- Backup and recovery operations involving the Vault are restricted to the mounted and authenticated state only, ensuring that backup passwords or secrets cannot be extracted in offline scenarios. This design makes "Harvest Now, Decrypt Later" attacks infeasible, as the Vault remains cryptographically and physically tied to its original secure execution environment.

3. Network Access and Exfiltration Protection

- Outgoing network traffic remains strictly controlled and policy-driven. For sensitive automation phases, temporary network blocking or user-confirmed release can be enforced to ensure that no unauthorized or automatic data transmissions occur. This flexible approach allows the

system to maintain a high level of security while avoiding unnecessary disruption to legitimate use cases where connectivity is essential.

- Users retain full control over provider trust relationships through a local whitelist, which can be modified or revoked at any time.

4. Authenticity and Integrity of WR Codes and Templates

- Every WR Code and automation template including provided context data is securely verified through a cryptographic hash.
- Templates and context without a valid signature or with altered structure are strictly not executed.
- Once a WR Code provider is whitelisted, it cannot alter automation templates without risking automatic delisting. Only explicitly trusted providers may be permitted to update templates when necessary.
- This restriction is particularly important for dynamically generated WR Codes, such as those displayed on digital screens or delivered over the internet, where the underlying automation templates are created dynamically. Such dynamic operations are only allowed for fully trusted and explicitly whitelisted providers.

- Even in these cases, every dynamically generated automation template can be securely logged on Solana, allowing users to verify and prove at any time that a specific automation was issued by a specific provider at a specific point in time.
- Importantly, any shared automation can be enhanced, adapted, or overwritten directly on the user's device. This flexibility ensures that workflows remain adaptable to specific needs, contexts, and preferences while maintaining security and control. Users are not locked into fixed automations and can modify or fine-tune processes to achieve the best possible outcome.
- Additionally, user-adapted workflows can be stored locally so that repetitive QR scans of the same automation will automatically use the adapted version instead of the original. This enables a continuous optimization of recurring processes while preserving full user control over the automation behavior.
- The integrity of every automation is documented through tamper-proof IOTA logs, providing forensic evidence in case of disputes.
- Importantly, any shared automation can be enhanced, adapted, or overwritten directly on the user's device. This flexibility ensures that workflows remain adaptable to specific needs, contexts, and preferences while maintaining security and control. Users are not locked into fixed automations and can modify or fine-tune processes to achieve the best possible outcome.

- Additionally, user-adapted workflows can be stored locally so that repetitive QR scans of the same automation will automatically use the adapted version instead of the original. This enables a continuous optimization of recurring processes while preserving full user control over the automation behavior.

5. Adaptive Trust Levels

To further strengthen trust and security, WR Code implementations may optionally support Zero-Knowledge Proof (ZKP) based qualification checks. This would allow automation templates to require verifiable but privacy-preserving proofs from users or devices before execution. Such proofs could demonstrate:

- Eligibility (e.g., minimum age or verified identity) without disclosing sensitive information,
- Compliance with security policies or certifications,
- Device integrity or vault status.

This optional qualification layer enhances the system's ability to meet regulatory requirements and supports high-trust applications without compromising user privacy or control.

6. WR CODE uses a multi-level security model to flexibly secure various use cases:

| Trust Level | Description | Examples |
|------------------------------|--|-----------------------------------|
| Level 0: Air-Gap Mode | No network, no PII transmission | Offline access control |
| Level 1: Manual-Only Mode | Non-sensitive automation only, no PII | Scooter sharing onboarding |
| Level 2: Trusted Mode | Automation after prior user approval, limited PII | Recurring services |
| Level 3: High Security Mode | Sensitive processes with mandatory biometric, hardware key or passphrase, ensuring that any release of PII requires explicit user confirmation | Contractual or IP-sensitive tasks |

7. Tamper Protection and Transparency

- All automations undergo a policy and template verification before execution.

- Outgoing network activities are blocked or explicitly controlled during sensitive phases, ensuring that PII is never prematurely exposed.
- By separating data visualization for non-sensitive content (via secure overlays) from actual data transmission and PII handling, WR CODE ensures that sensitive information cannot be silently extracted by malicious templates or external scripts.
- Users receive visible control options and can review, track, and revoke automated decisions at any time.

Governance Considerations (Informal)

WR Code is designed as an open and community-driven standard that defines how QR automation templates can be created, verified, and transparently shared. A **WR Code** contains a **signed payload** that embeds only **lightweight pointers** (such as template IDs, manifest references, or policy selectors) — not AI instructions or logic itself.

The actual automation templates are **externally hosted**, **tamper-proof**, and **publicly auditable**, with verification anchored on **Solana** via WRStamp.

When scanned, the WR Code triggers execution either **locally on the user's device** or via a **remote or directly connected orchestrator**, depending on system capabilities and trust policies.

This hybrid design preserves the **portability of QR technology** while enforcing **cryptographic integrity, template and context consistency**, and **issuer accountability**.

To support integrity, trust, and transparency, wricode.org serves as a public validation platform where organizations authenticate themselves and register their WR Code automation templates. This process allows templates and context to be made publicly visible, tamper-proof, and immutably logged—using Solana—ensuring that:

- Templates can be proven to belong to a specific verified organization.
- Template code remains public, transparent, and resistant to manipulation.
- End users can verify the legitimacy of automation templates before use, and in case of fraud attempts, all interactions can be forensically traced and proven.

This approach allows users to freely choose how they interact with WR Codes.

Future governance may evolve to further define the fair use of the WR Code ecosystem as adoption grows.

WR Code is designed as an open and community-driven standard. While the system encourages broad adoption in both closed and open environments, it is intended to remain associated with transparent, privacy-preserving automation principles as implemented in **WR CODE**.

Future governance may evolve to further define the fair use of the WR Code ecosystem as adoption grows.

Conclusion

Looking ahead, **WR Codes** may also gain relevance for emerging technologies such as AR glasses and wearable devices, where quick, secure, and standardized automation triggers will become increasingly valuable. The open nature of the **WR Code** standard ensures that it can flexibly adapt to such future use cases while maintaining a strong focus on privacy, interoperability, and user control.

This security concept ensures:

- Maximum user control
- Verifiable integrity of all automations and QR interactions
- Protection against auto-pull, manipulation, and exfiltration

WR CODE not only protects data but also empowers users with digital self-determination against modern threats.

Open Use, Verifiable Trust, Confidentiality, and Secure Interoperability with WR Codes

WR Codes are based on the globally recognized QR Code Model 40 standard, part of the international ISO/IEC 18004 specification. While "QR Code" is a registered trademark of Denso Wave

Inc., the underlying technology has been made freely available for both commercial and private use worldwide.

WR Codes do not change the QR Code standard; they define a verifiable way to use standard QR Codes as triggers for flexible automations. Individuals and organizations can create and use WR Codes entirely within private or internal infrastructures. When desired, publishers can release WR Codes for public use, enabling fast, cost-effective automation while preserving privacy and full control over internal processes.

Building Trust through Verifiable Automation

In cases where WR Codes are used in public-facing services—for example, customer onboarding, mobility apps, or hospitality—trust, transparency, and legal defensibility become essential. To meet these needs, WR Codes can be optionally registered at wrcode.org:

- Each WR Code is linked to a tamper-proof cryptographic hash stored immutably on decentralized ledgers such as Solana or Optimism.
- This allows end users to verify that the automation triggered by the WR Code matches the registered, untampered template and context.
- The original automation template must also be locally accessible and unmodified at the time of execution, ensuring that both hash verification and content verification are possible.

This dual verification ensures that users can:

- Confirm the authenticity of the automation,
- Audit and prove the exact process that was executed,
- Build trust in customer-facing services while ensuring compliance with legal and security standards.

The wrcode.org registry is centrally governed to enforce ethical use, prevent abuse, and protect the integrity of the system. Registrations may be declined or revoked if policies are violated.

Confidential Automations: Optional Disclosure, Always Verifiable

In high-risk operations—such as sensitive industries, regulated environments, or security-critical contexts—organizations may choose to keep their automation templates undisclosed while still maintaining full verifiability:

- The cryptographic hash of the automation is still registered,
- The original automation template remains local and private, yet can be verified against the hash.

This ensures that while the content of the automation remains confidential, its execution integrity is provable. End users interacting with such WR Codes are informed when a process is undisclosed and can decide whether to proceed. This preserves both privacy and user choice.

Full Interoperability and User-Controlled Data: The PII Vault (Dual Use)

For advanced use cases involving cross-device continuity, personal data reuse, and AI-powered automation, the Dual Use License introduces full interoperability through the WR CODE Orchestrator and QRGiraffe tool.

At the heart of this system is the PII Vault—but critically:

- The PII Vault (**WRVault**) belongs to the user, not to the publisher of the WR Code.
- The publisher (e.g., hotel, mobility service) may design WR Codes that request data to trigger specific automations.
- The user holds and controls this data inside their own private PII Vault, which functions as a kind of personal key to public automations.

The WRVault allows:

- Customizable, structured data storage—identity, preferences, transaction history, or any user-defined information,
- Data to be shared only when the user consents, with full transparency and security,
- **Context-Aware Automations with Dynamic WR Code Generation**

Public WR Codes can dynamically adapt not only the triggered automation but also the visual representation of the WR Code itself—based on context data stored in the user's private vault or derived from non-PII environmental signals. In selected scenarios, context can influence both the execution **and** the generation of the WR Code in a continuous feedback loop. For example, a user wearing AR glasses might see a WR Code overlay that is generated in real time based on non-personal contextual factors such as personal preferences, time, or device state. This enables highly personalized, adaptive interactions without exposing sensitive information. The dynamically generated WR Code can be selected from a predefined set of registered templates or fully customized to match the specific situation. All generated WR Codes remain **tamper-proof and verifiable** through cryptographic methods, ensuring transparent and secure automation triggers. Where appropriate, dynamic WR Codes can adhere to emerging **standards or ethical guidelines**, particularly in high-risk or sensitive environments, to maintain public trust and prevent misuse. This concept is currently intended for **research purposes**, exploring how context-aware visual codes could enable seamless, privacy-preserving automation across devices, AR systems, and connected environments.

This model ensures:

- Privacy-first interoperability,
- Frictionless onboarding without repetitive data entry,
- Security and legal compliance at every interaction point.

Private Use vs. Ecosystem Integration

Organizations that prefer to remain fully private can still use WR Codes within closed, self-contained systems without registering at wrcode.org. After all, WR Codes are only QR Codes that follow a specific standard guideline. These individual setups may involve internally managed automation templates and even private equivalents of the PII Vault.

However, such systems will not be interoperable with the global WR Code ecosystem, nor will they benefit from public trust, cross-device continuity, or ecosystem-driven AI orchestration.

Convenience in the Hotel with WRCode



**Easy
Check-In**



**Useful
Room Info**



**Order with
WRCode**



**Quick
Check-out**

QR Code technology has become a universal tool for bridging physical and digital experiences. Its simplicity, accessibility, and global adoption have made it indispensable across industries. While QR Codes themselves are technically harmless, there have been instances where malicious actors have used QR Codes to direct users to harmful websites or fraudulent services, leading to device infections, phishing attacks, or unauthorized data collection. These incidents underline the need for a secure, transparent framework for any future use of QR Codes in triggering trusted automations.

The **WRCode** represents this next step: a secure, standardized approach to trusted automation triggers, defined and governed by wrcode.org.

Defining the **WRCode**

While QR-triggered automation existed in fragmented or proprietary forms, **WRCode**, invented by **Oscar Schreyer** and published on wrcode.org, is the **first fully verifiable system** that transforms QR-based interactions into a **secure, standardized automation layer**.

This invention combines:

- Tamper-proof, publicly auditable automation templates, context and Source-available code libraries and tools , hosted transparently
- Cryptographic runtime validation , anchored on Solana
- Cryptographic optional runtime logging , anchored on IOTA

- A user-controlled **WRVault** for secure, privacy-preserving personalization
- **Customizability:** AI instructions can be enhanced or modified locally, and publisher context (from WRCode.org) can be augmented with the user's own on-prem context and private memory.
- And a strict **governance model** that enforces **publisher verification, template and context integrity, and policy compliance**
- On-prem automation with hybrid or cloud-based LLM support
- Open-weight LLMs are tailored to the orchestrator and can be further fine-tuned via a user-friendly UI or as a personalization service.
- Workflows, AI outputs, and tools are organized into multiple grids across separate browser tabs, allowing instant switching and smooth handling of even the most complex, multi-faceted AI tasks.

WRCode establishes a new standard for **trustable, on-demand digital actions** — portable, transparent, and privacy-respecting by design.

What Makes WRCode Unique

A WRCode remains fundamentally a standard **QR Code** conforming to the ISO/IEC 18004 specification — the QR technology itself is unchanged.

What sets a WRCode apart is its integration into a **trust-first automation framework** , defined by:

- Strict security and operational standards maintained by wrcode.org.
- Automation templates, context and policy manifests that are publicly hosted on wrcode.org , versioned, and tamper-proofed through WRStamp signatures.
- Anchoring on Solana and IOTA , providing cryptographic timestamping and fast, decentralized validation.
- WRVault integration, enabling secure, user-controlled personalization without exposing PII.
- A verifiable ecosystem designed for issuer accountability , public auditability , and privacy-respecting customer-facing automation .

In summary: while every WRCode is technically a QR Code, only those QR Codes that conform to the WRCode guidelines qualify as WRCode.

The WRCode Security Framework

Access to PII stored within the Vault is strictly denied unless the **WRCode** automation is verified in real time through an active internet connection. This is essential to validate the tamper-proof zero-knowledge proofs of the software stack, the **WRCode** issuer, and the device fingerprint anchored on the Solana blockchain. Without this verification, no sensitive automation involving PII can be executed.

In scenarios where users scan standard QR Codes or **WRCode** automations while offline, the system will operate in a “non-sensitive” mode: QR Codes can still be scanned and analyzed, and informational content can be displayed, but no AI automation instructions from QR Codes will be triggered, and no PII stored in the Vault will be accessed or released.

This approach ensures that **WRCode**-compliant applications remain fully functional for general QR Code scanning and offer safe, controlled environments for analyzing standard QR Codes without compromising the strict security and privacy framework of **WRCode** automations. Full **WRCode** functionality, including AI automation execution and access to sensitive data, is only permitted once the system confirms the authenticity of the entire software and identity chain through **zero-knowledge proof online verification**.

To ensure authenticity and prevent misuse, all Vault applications and **WRCode**-compliant software must include a **mandatory WRCode Validator**. This built-in validator checks whether a scanned QR

Code conforms to the **WRCode** standard by verifying its registration, the anchoring of its automation template, and the validity of the issuer identity through wrcode.org.

Only QR Codes that successfully pass this validation are permitted to trigger sensitive automations or request access to personal data stored in the Vault. Non-**WRCode** QR Codes may still be processed by compliant software, such as for analyzing offers or reading non-automated content, but these cases must never access or unlock Vault-stored PII.

This mandatory **WRCode** validation ensures that users and organizations can independently verify whether a code is a genuine **WRCode**, providing a consistent layer of trust and security across all implementations.

The **WRCode** standard is built upon a clearly defined set of security and operational guidelines established by wrcode.org, which together ensure the integrity, trustworthiness, and transparency of all **WRCode**-based automations. These guidelines include:

1. **Mandatory Verifiability:** All **WRCode** automations—whether public-facing or internal—must be verifiable through wrcode.org. For customer-facing **WRCode** automations, the corresponding automation templates must also be transparently published unless there is a justified reason for confidentiality. In cases where templates remain confidential, their existence must still be provably verifiable through cryptographic anchoring on the **Solana blockchain**. Additionally, issuers opting for confidentiality must provide a **public**

statement through wrcode.org explaining the nature of the confidentiality, ensuring that users and stakeholders are aware of the verifiable, yet undisclosed, status of the automation template.

2. **Tamper-Proof Templates & Context:** Every automation template and public context package is cryptographically hashed, versioned, and anchored on Solana to enable immutability and tamper resistance. For private/on-prem context, only the hash (and minimal metadata) is anchored—content stays local—so integrity is verifiable without disclosure.
 3. **Verified Issuer Identities:** Organizations issuing WRCode automations must undergo identity verification conducted through wrcode.org. The verified identity is then cryptographically hashed and anchored on the **Solana DLT**, providing tamper-proof verification of the issuer while safeguarding sensitive identity details. This ensures that only authorized publishers can issue trusted automations and that the authenticity of the issuer can always be validated.
- #### 4. Secure Vault Software

The Vault is a local, on-device password and identity manager used within the OpenGiraffe/WRCode ecosystem.

It stores credentials, identities, session keys, and other sensitive data in encrypted form on the user's device.

The Vault operates exclusively on the local machine.

Sensitive data never leaves the device unless the user explicitly approves an exchange.

The Vault can securely exchange data with OpenGiraffe-compatible applications that follow the WRCode guidelines published at wrcode.org.

These applications use standardized protocols to request access, and the user approves each interaction through the Vault's interface.

All sensitive records remain isolated inside the Vault, and only WRCode-compliant applications are able to request or receive data. Unknown or non-compliant software cannot access any information.

This design provides a controlled trust model where:

- sensitive information stays encrypted on-device,
- user approvals govern all data exchanges, and
- interoperability is limited to verified, OpenGiraffe-compatible applications.

Future Execution Layer: QRGiraffe

The forthcoming QRGiraffe orchestration environment will leverage the full potential of the WR CODE AI Agent Orchestrator to extend and enhance the capabilities of WRCode-based automations.

QRGiraffe will not be required for WRCode adoption but will serve as an example of how to build advanced, AI-driven automation experiences on top of the WRCode standard.

Timestamped, Open Innovation

The WRCode standard has been:

- Publicly documented and cryptographically timestamped using OpenTimestamps on the Bitcoin blockchain .
- Released under an open, community-driven model to ensure transparency and verifiability.

This highlights the authenticity, originality, and trustworthiness of the WRCode framework.

The Importance of WRCode

Without a secure standard, QR-based automations would remain vulnerable to:

- Tampering and manipulation
- Spoofed automations and phishing
- Lack of issuer accountability
- Privacy breaches

The WRCode standard addresses these risks through:

- Verified, tamper-proof automation triggers
- Verified, tamper-proof publisher context
- Transparent validation and accountability
- Strict Vault and privacy safeguards

Conclusion

The WRCode establishes a **trusted, transparent, and verifiable global standard** for QR-triggered automation:

- Only QR Codes that meet wrcode.org standards are recognized as WRCode.
- All public WRCode automations must be **registered, anchored, and publicly verifiable**.
- Vault software must be **secured, verifiable, and anchored on Solana**.

By following these principles, WRCode empowers organizations and individuals to adopt QR-triggered automation in a manner that is secure, accountable, and privacy-respecting.

Hotel Check-In Using WRCode: Practical Example

Scenario Overview

A guest arrives at a hotel without a prior booking. In the lobby, a **digital screen displays a dynamically generated WRCode** that includes all relevant booking details, such as room availability, pricing, and optional services. The guest scans the **WRCode** using their smartphone or another device equipped with a **WRCode**-compliant Vault.

This touchless process allows the guest to instantly review the booking information directly on their device. With a single secure confirmation in the Vault, the booking, payment, and check-in process is executed automatically. The **WRCode** ensures that all automations, including PII handling and transaction processing, are completed securely and transparently without the need to interact with hotel staff.

Step 1: Immediate Check-In

- The **WRCode** triggers a PII consent screen in the Vault.
- The guest approves, and the Vault validates the **WRCode** through **zero-knowledge proof online verification** (hash-only lookup on Solana).
- The guest's preferences—such as room temperature, pillow type, or loyalty rewards—are preconfigured by the guest using the hotel's **WRCode** generator available on the hotel's

website. Through fine-grained slider settings and service choices, the guest creates a personalized **WRCode** in advance, which is securely stored in the Vault as non-sensitive external data specifically for later use in automated booking and check-in.

Step 2: Arrival & Access

- Upon arrival at the hotel room, the guest uses the same **WRCode** to unlock the door.
- All preferences are pre-applied without additional steps.
- The Vault ensures that no sensitive data is shared unless validated and approved.

Step 3: Hotel Services and Stay

- The guest can use additional **WRCode**-enabled services:
 - Order drinks or food by scanning **WR Codes** in the room or bar, with charges linked to the room or paid directly via a secure Vault-stored payment method.
 - Book spa appointments or restaurant tables by scanning other **WR Codes** around the hotel.
 - Connect to the **WR CODE Orchestrator** on a laptop or tablet to access personalized suggestions: upcoming events, late checkout offers, or tailored experiences.

- The AI automation templates embedded in the WRCode are executed automatically to optimize the stay.
- An intelligent WR CODE optimization layer runs in parallel, checking for better options or dynamic recommendations based on the guest's preferences, unless toggled off.

Step 4: Check-Out

- Check-out is fully automated via WRCode or the orchestrator.
- Guests receive digital invoices, loyalty point updates, and follow-up service offers.

Security and Privacy

- All WRCode automations are executed only after zero-knowledge proof validation using Solana anchors.
- No sensitive PII is ever shared or processed without full consent.
- Unverified WR Codes are discarded automatically without execution.
- Optional transaction logs for orders or service confirmations are stored securely within the Vault for verifiability but without exposing personal details.

Offline Use

- If scanned offline, a **WRCode** behaves like a normal QR Code, displaying basic non-sensitive information.
- If a **WRCode** is detected but cannot be verified immediately, it is securely stored separately from PII until validation is possible. If it fails validation, it is deleted without execution.



WRCode represents a proposed standard for secure, verifiable QR Code-triggered automation. Building on familiar QR Code technology, WRCode introduces strict security guidelines, account-based identity anchoring, and trustless user control while maintaining accessibility and privacy. This paper outlines additional security, resilience, and automation concepts that complete the WRCode ecosystem.

Core Principles

- **Mandatory Verification for WRCode Publishers:** Publishers (merchants, service providers, organizations) must register at WRCode.org and pass a stricter verification process: domain ownership via DNS challenge plus email verification. The resulting verification attestation (e.g., hashed metadata) is anchored on Solana for a tamper-proof record.
- **Mandatory Verification for Users:** All end users must complete email verification before interacting with WRCode automations. A hashed proof of verification is likewise anchored on Solana. Additional identity (e.g., OIDC/KYC/wallet) is optional and enforced only when a specific template or regulation requires it.

Optional Identity Anchoring for Users

Benefits of email identity anchoring:

- Unlock faster service in high-trust scenarios (e.g., hotel check-in, car rental).

- Enable automated invoicing, business travel expense management, and personalization.
- Facilitate optional Vault recovery and revocation in case of device loss, ensuring users can securely disable lost Vaults and restore their identity on a new device.

The identity anchor is derived exclusively from the user's email address, cryptographically hashed and securely anchored on Solana. No other personal information is collected, exposed, or shared with **WRCode** providers, and the email remains private. It is neither published nor shared unless required by lawful authority. **WRCode.org**'s role is limited to verifying the authenticity and integrity of the anchor without the ability to access or disclose the user's identity.

Resilience: Trustless Vault Backup and Revocation

To ensure continuity and user safety in case of device loss, **WRCode** introduces an optional backup and recovery feature based on trustless security by design:

- Encrypted Backup: Vaults are backed up locally or optionally in encrypted cloud storage, always tied to the user's **WRCode** account (email address). A cryptographic hash of the original device's hardware fingerprint must be saved by the user (on a USB stick, hardware key, or secure medium). Restoration requires:
 1. The verified email account.
 2. The hardware fingerprint from the original device.

3. The user-defined passphrase.

- Mandatory Multi-Factor Recovery: A portion of the recovery data is stored as an account-bound hash fragment. Without all three factors—account access, device fingerprint, and passphrase—the Vault remains permanently inaccessible. Even if the Vault container is copied, it cannot be decrypted without the full set of credentials.
- Tamper-Proof Revocation: In case of device loss, users can revoke the old Vault and issue a new Vault bound to new hardware, ensuring continuity without compromising security. This recovery feature is offered as part of **WRCode's** optional premium service tier.
- Self-Controlled: Neither **WRCode.org** nor any third party holds decryption capability. All sensitive operations remain exclusively under user control.

This layered approach ensures that Vaults cannot be stolen, copied, or misused—upholding true trustless security.

Contextual Automation: Local AI Assistance

To enhance usability while preserving privacy, **WRCode** supports the use of lightweight, device-local language models (LLMs), securely stored and executed within the Vault:

- Users can issue simple voice or text commands (e.g., "Late checkout" or "Order coffee") processed entirely on-device.

- The local LLM assists in mapping user preferences to automation templates without exposing data to external servers.

Paperless Payments. Automated Outcomes.



Printing receipts isn't good for the environment. Thermal paper often uses chemicals that put people and the environment at risk.

Scenario: Checkout Reinvented—Private, Paperless, Automated

Lisa visits her favorite fashion store. Instead of receiving a printed receipt, the checkout screen displays a **WRCode**—a secure, tamper-proof QR Code linked to a verifiable automation template.

Step 1: Scan, Pay, and Share Preferences in One Flow

- Lisa scans the **WRCode** using her **WRCode**-compatible app.
- The app:
 - Retrieves Lisa's secure identity anchor (a cryptographically hashed zero-knowledge proof anchor stored on **Solana**).
 - Uses her PII Vault, where her preferred payment method (e.g., credit card, Apple Pay, crypto wallet, QR Pay or Cash) is already securely stored.
 - Executes the payment directly from the Vault—no need to re-enter any sensitive information. Simple confirmation per fingerprint or FaceID.

Lisa's payment credentials are securely provided to the **WRCode-verified** merchant and the transaction is logged tamper-proof on **IOTA** for future audits.

The store sees only the payment data necessary for transaction execution—nothing more.

Step 2: Automated Receipt & Personalization (Optional)

- Immediately after checkout, a second WRCode appears.
- This **WRCode** is:
 - Dynamically tailored to Lisa's pre-shared preferences (e.g., she uses Lexoffice for taxes).
 - Securely linked to the transaction and ready to trigger automation.

Lisa skips scanning this second code in the shop—it's optional.

Step 3: Deferred Automation via Orchestrator

- At home, Lisa opens her Orchestrator App.
- The app:
 - Recognizes her WRTransaction, stored securely in the Vault alongside the payment proof (no paper, no email).
 - Displays the personalized second **WRCode** linked to her earlier payment.
- Lisa scans this **WRCode**—because she had securely shared some basic preferences earlier from her Vault, the merchant had already dynamically tailored this second **WRCode** during checkout and attached it to the transaction:

- The invoice is enhanced with her WRVault PII (e.g., name, address, tax number).
 - Automatically submitted to Lexoffice, WISO Tax, Taxman or other integrated systems based on the taylored automation template.
 - The entire process is anchored on IOTA—verifiable, timestamped, and tamper-proof.
-

Key Innovation: Payment Data Stored and Controlled in the Vault

- Users store multiple payment options in the WRVault:
 - Credit/Debit Cards (tokenized or via secure keychain)
 - QR Pay (e.g., Alipay, WeChat Pay, SEPA QR, Girocode, PayPal QR)
 - Crypto wallets (optional future integration)
- The payment is initiated via the Vault but—for payment execution—data is securely shared with the WRCode-verified merchant.
- The transaction details are immutably logged on IOTA, ensuring legal defensibility, privacy, and compliance.

- Even contactless in-store payments can be triggered by WRCode → Vault → Payment in seconds.
-

Strategic Messaging: WRCode as the Missing Link Between Payment, Automation, and Privacy

Tagline Ideas:

1. "One Scan. One Pay. One Click to Automate Your Life."
 2. "Receipts That Write Themselves. Taxes That File Themselves."
 3. "No Paper. No Apps. Just You, Your Vault, and the Power to Automate."
-

Environmental Impact: Why Paperless Matters

- Thermal Paper Receipts Are Harmful: Most printed receipts use thermal paper coated with chemicals such as BPA or BPS, which are known to be harmful to both human health and the environment.

- Environmental Waste: Billions of paper receipts are printed every year, generating unnecessary waste and contributing to deforestation, pollution, and landfill overflow.
- **WRCode Eliminates the Need for Printed Receipts:** By moving to digital, verifiable receipts via **WRCode**, merchants and consumers help reduce chemical exposure, save trees, and cut waste.

Bonus Features to Market:

- Budget Automation: **WRCode** preferences could link expenses directly to personal budgets or savings plans.
- No App Lock-In: The open standard allows multiple wallet and vault apps to co-exist.
- True Privacy: Payment data is shared only with verified merchants and recorded immutably for compliance—never with third parties, only with **WRCode-verified WRCode** providers as necessary for the automation.

This is not just payment automation—it's identity + payment + compliance + automation in one frictionless moment.

Introducing WRCode: A Trustless Automation Framework for Payment, Identity, and Loyalty

The digital landscape is evolving rapidly, and businesses require flexible, secure, and privacy-first solutions that adapt to a variety of use cases. **WRCode** represents a next-generation framework that enables merchants, service providers, and users to interact seamlessly through decentralized automation. Unlike traditional systems that rely on centralized databases, **WRCode** offers a modular, trustless approach that gives control back to users while enhancing business opportunities.

WRCode is free to use for all end users, including those who wish to benefit from paperless receipts, rewards, and basic bonuses—without the need for any registration. Some advanced features—such as quick onboarding, fast-lane check-ins, formal invoicing with required business information, or multi-device vault recovery—may require an optional lightweight registration. This registration ensures that both compliance and advanced convenience features can be offered while maintaining **WRCode's** core principle of user control and privacy-first design.

The Core: **WRCode.org**

At the heart of the system lies **WRCode.org** — the open protocol that powers all WR-branded modules. It serves as the secure, standardized foundation that ensures interoperability, privacy, and flexibility across various industries.

Key Modules Explained

WRPay — Privacy-First Payment

WRPay enables fast, flexible payment solutions that respect user privacy. Whether used for digital transactions or as a paperless complement to cash payments, WRPay eliminates unnecessary data exposure while offering merchants new automation possibilities.

Benefits over traditional payment systems:

- Only absolutely necessary data shared
 - Paperless receipts, eco-friendly
 - Optional backend automation triggers for tailored experiences
-

WRCollect — Loyalty & Rewards Reinvented

WRCollect allows businesses to offer dynamic, customizable loyalty programs without intrusive data collection. Customers "collect" benefits, such as discounts, gifts, or exclusive offers, by interacting

with WR-enabled merchants. Unlike traditional systems that centralize loyalty data, **WRCollect** stores information locally or encrypted, keeping user preferences private.

For merchants, **WRCollect** provides a powerful incentive tool: it enables creative and personalized rewards that go beyond simple points or generic discounts. For example, a customer who spends over a certain amount might receive a free coffee on their next visit or unlock exclusive deals—entirely automated through **WRCode** without complex backend systems. This could mean: "Collect a bottle of wine on your next shopping trip" or "Unlock a personalized discount after your third visit," with all interactions remaining privacy-friendly and seamless.

Benefits over traditional loyalty systems:

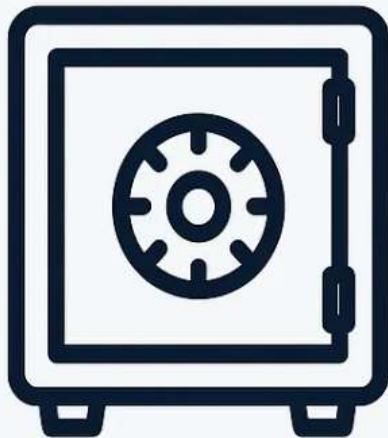
- No central data mining
- Interoperable across merchants
- Customizable benefits and creative incentives that drive repeat business
- Only absolutely necessary data shared
- Paperless receipts, eco-friendly
- Optional backend automation triggers for tailored experiences

WRPass — Decentralized Identity & Access

WRPass replaces conventional logins and verification methods with a secure, WR-based identity layer. Acting as an on-premise, **source-available** password manager and verification tool, **WRPass** enables all types of identity proofs—whether zero-knowledge-based or traditional—depending on the specific use case. Users retain full control over their credentials through a local vault, allowing trustless, secure verification for both digital and physical access while ensuring sensitive data never leaves their device unless explicitly shared.

Benefits over traditional identity systems:

- No passwords stored in the cloud
 - Verifications without revealing unnecessary information
 - Suitable for digital and physical access control
4. WRVault — Secure Data Storage



Account Login

Recovery Phrase

Recovery Hash



Recovery

WRVault serves as a private, user-controlled storage space for sensitive data, preferences, or automation keys. Each vault is cryptographically bound to the user's **WRCode** account, ensuring that only the rightful account holder can manage, revoke, or migrate vaults. Upon creation, users generate a unique recovery hash, which is stored securely offline (e.g., on a USB stick or printed backup). This recovery hash, combined with account access, is required to trigger any vault revocation or migration process.

To maintain strict security, **WRCode.org** does not store recovery hashes, passphrases, or vault keys. Only a non-reversible public fingerprint of the vault is optionally stored to allow users to reference and manage their vaults securely. Revocation or migration requests must be authorized both through account login and the presentation of the user-held recovery proof, ensuring that no attacker or insider can trigger revocations without the user's active involvement.

If both the recovery hash and any optional passphrase are lost, the vault becomes unrecoverable by design—ensuring true zero-trust security without any central killswitch. Even in the unlikely event of a **WRCode.org** server breach, attackers cannot mass-revoke or access vaults (vaults are stored locally anyway), as critical recovery materials are never centrally stored and remain fully under user control.

For advanced users who wish to benefit from more powerful cloud-based AI services, **WRVault** offers configurable options where users can explicitly authorize selected data to be processed externally.

This flexibility preserves the core principles of trustless design and full user control, while allowing tailored experiences for those who opt in.

Both the original structured vault data and any embedded semantic layer coexist on-premise or within a user-selected architecture, maintaining security while unlocking the power of AI-enhanced workflows.

Compliance and Invoicing in WRCode

WRCode enables fully paperless, anonymous invoicing for everyday purchases by default. In most cases, this is legally sufficient and respects the user's privacy.

When merchants need to issue formal invoices that include regulated data (such as tax numbers or business details), the WR application they use must include the necessary business logic to prompt users for optional data disclosure.

Users can choose to:

- Share a zero-knowledge proof identity (email + token) to receive compliant digital invoices.
- Share unmasked personal information (PII) if required for bookkeeping or legal reasons.

- Or receive an anonymous invoice and add necessary details manually later if they wish to claim business expenses or tax benefits.

The control remains entirely with the user, while merchants retain the ability to meet regulatory needs within their own WR applications.

WRPay – Technical Flow Description

WRPay enables flexible, real-time payments through WRCode orchestration, allowing users to initiate payment workflows by simply scanning a code — with or without prior login, registration, or manual setup. All workflows are defined in **signed, verifiable templates** , which can be checked through **wrcode.org** or **trusted local runtimes** . While actual payment execution depends on the provider, the surrounding logic — including user consent, data flow, and orchestration steps — is **cryptographically auditable and tamper-resistant** .

To support broader adoption, WRCode will provide a library of vetted templates and code modules , enabling **verified developers** to build trusted applications with ease. These building blocks support everything from simple automation to complex, multi-agent logic — across both open and proprietary ecosystems.

The goal is to make **secure automation as flexible, composable, and auditable as possible** — without sacrificing user control or system integrity. WRCode combines **verifiable trust with developer freedom**, creating a foundation where powerful workflows can be built and deployed safely, regardless of scale or use case.

The strength of **WRPay** lies in the flexibility of **WRCode**. A **WRCode** can trigger not only static templates but also dynamic, provider-controlled workflows that integrate with apps, backends, or payment systems. This includes:

1. At **checkout**, the vendor system (e.g. cash register or terminal, website) generates a **WRCode** and displays it on the screen.
 - o The **WRCode** contains a reference to a **TemplateMap**, which defines the orchestration logic, and includes a session-specific nonce and offer metadata (amount, timestamp, etc.).
2. The **user scans the WRCode** with their **WRVault**. This triggers:
 - o Business logic is **not executed or interpreted** at this stage. The vault performs only validation and orchestration setup based on publicly anchored instructions.
 - o The downloaded templates may contain device-specific AI orchestration instructions for multi-agent flows — for example, automated bookkeeping or contract fulfillment. All

such logic must be publicly defined, hash-verified, and Solana-anchored via the TemplateMap. These templates are downloaded from wricode.org or a trusted third-party source and validated before any execution occurs.

- A **live, uncached verification** of the vendor via Solana, checking:
 - That the vendor is legitimate and registered
 - That the TemplateMap, templates and context are anchored and unmodified
 - That the session data has not been reused or altered
- 3. If all verifications succeed, the **user sees the full transaction details** (price, vendor, optional metadata) on their device. A **biometric approval prompt** (e.g. fingerprint, Face ID) appears.
- 4. Upon biometric confirmation:
 - The **transaction is logged on Solana** .
 - The **vendor system stores a session token** , which is cryptographically matched to:
 - A **hardware- and software-bound token** generated on the user's device

This token:

- Is worthless on its own — it **requires a live Solana-authenticated match**

- The token stored on the vendor machine is worthless on its own — it cannot be copied, reused, or exploited without the matching root token from the user's WRVault.

5. Repeat visits are optimized:

- When the same user returns to the same vendor, **auto-pairing** with the terminal occurs in the background.
- During the biometric approval (~800ms), the system re-verifies the session and tokens via Solana in parallel.
- Because **TemplateMap, templates and context are minimal in size**, even slight network delays don't disrupt the flow.

6. Autopairing and Session Resumption

On repeat visits, autopairing allows the WRVault to securely reconnect to a previously visited vendor system without needing to scan a WRCode again. This works as follows:

- During the first transaction, the vendor stores a fingerprint of the user vault (e.g. hash of vault ID and session UUID).
- On future visits, the vendor device emits a pairing request (via BLE beacon, NFC, or secure WebSocket) containing its vendor hash.

- The WRVault detects this and compares it with stored known vendors.
- If it matches, the vault silently verifies the vendor via Solana and preloads the TemplateMap.
- As the user approaches or interacts, the biometric prompt is shown.
- All token handshakes and verifications happen in the background.

Autopairing never bypasses user approval. No session can be reused, and all flows remain cryptographically verifiable in real time.

7. wrcode.org operates its own Solana and IOTA nodes, with fallback to public infrastructure. Real-time validation is exclusively handled via high-performance blockchains such as Solana and Optimism. IOTA is used only for optional, long-term archival anchoring and not involved in time-critical verification processes..

8. Mobile App Invocation (No WRCode Scan)

If the user triggers a WRPay transaction from within a mobile app (e.g. a store or e-commerce app), no QR scan is needed. Instead:

- The app passes a reference to the **TemplateMap**, session metadata, and offer details to the WRVault via deep link, intent, or embedded SDK.

- The WRVault downloads the TemplateMap and templates, verifies hashes against Solana, and prompts the user for biometric approval.
- The rest of the flow — token generation, pairing via Solana, and logging — follows the exact same trustless protocol.

To maintain trustless character:

- No business logic or executable content is allowed to be passed from the app
- TemplateMap and all templates must be public and hash-anchored
- Only pre-approved metadata is accepted by the vault

9. Distributed Template Hosting with Hash Verification

To reduce load and latency, publishers may host the TemplateMap and templates themselves.

However, before execution:

- The WRVault downloads the templates and computes their SHA-256 hashes
- These are compared against Solana-anchored hash records published via wrcode.org
- If any mismatch is found, execution is aborted

This setup enables:

- Decentralized hosting (optional for applications that want to help reduce latency even more)
- Transparent, tamper-proof validation
- Use of Merkle roots for improved verification performance
- Compression support, as long as the final uncompressed content is verified

This ensures consistent security while supporting scale, redundancy, and performance optimization.

WRCode is increadible flexible

- Automatic onboarding into apps or services
- Conditional release of payment and identity data
- Seamless integration with proprietary business logic
- Strong focus on users giving them real-time AI optimization support

GeoFencing with WR Codes

High-Impact Use Cases



Public
Transport



Hospital



Airport



Geofenced WRCode: High-Impact Use Cases That Scale

WRCode is more than just a QR code — it serves as a secure, decentralized trigger for automation. Each WRCode references a predefined AI template that is executed locally whenever possible, using lightweight models running on the user's smartphone, AR glasses, or wearable. Once scanned, the user is redirected to the official app in guest mode, where minimal, intent-based instructions are delivered — potentially by a local AI agent, but more commonly as structured prompts from the AI template itself— which are then processed by a small on-device LLM. For example, "*Where do you want to go?*" might be the instruction delivered via voice or touch. In this case, the template does not contain code but descriptive AI instructions that guide the local model to generate context-aware input using local data sources such as from the WRVault, geofencing context, or — when permitted — limited or full PII like health or identity data.

User data, including preferences, accessibility settings, and optional PII, is securely stored in a local WRVault — typically on the phone or even a WR-connected device such as a smartwatch. These personal edge devices may also carry context-relevant data, such as health information, that can be accessed automatically if permitted. The system enables device pairing across a private mesh, allowing the user's own devices to contribute securely to a local context session.

When required, WRConnect establishes a secure link between the user's personal device mesh and external IoT infrastructure, such as public displays, EV chargers, or hospital terminals. The provider

never sees identity information — it only connects to the active session via a short-lived, scoped interface. As an added incentive, specific providers may offer **WRCollect** items as rewards. For instance, a user who visits daily for 7 consecutive days could automatically receive a free coffee — triggered and validated entirely through the **WRCode** system without needing user identification or manual check-ins. Even if the user does not actively scan the **WRCode** each day, geofencing can register presence and anchor proof of visit in a tamper-proof way on the Solana blockchain. The **WRCode** only needs to be scanned once to initialize the reward logic. Subsequent visits can be anonymously verified through geofencing alone, using zero-knowledge proofs and local validation — without requiring further scans or revealing any personal data. This allows even the first visit to trigger the process securely and privately.

Geofencing further enhances this model by ensuring that automations are activated only in precise physical locations. The GPS signal never leaves the device; it's used only to validate the location context locally. This allows the same **WRCode** to behave differently depending on where it is scanned, without any user tracking or centralized identity resolution.

Additionally, a background optimization layer may run silently on the device to support more complex workflows utilizing more powerful cloud-based ai chatbots or a connected local orchestrator in the own network. This layer interprets the current situation and provides enhanced support when needed — for example, during emergencies or highly dynamic interactions. In most everyday scenarios, the standard **WRCode** execution using local context and local small llms on

mobile devices is entirely sufficient. When toggled on, the optimization layer can be opened or closed at any time without disrupting the active session, allowing it to function seamlessly alongside **WRCode** connections without requiring the user to close or restart the connected app.

This architecture allows for highly adaptive, deeply personalized workflows that scale across sectors — while keeping privacy, control, and execution fully in the hands of the user.

This background optimization uses the available user input, location context, and the AI template logic to enhance the response. While **WRCode** and **WR CODE** are not against cloud AI, it is important to note that such systems may access user identifiers like email addresses depending on the provider. High-risk individuals should be aware of this and are advised to keep cloud-based optimization disabled in sensitive situations to avoid leaking PII or other private data.

Privacy and control stay fully on the user's side. This model enables frictionless, meaningful automation at real-world scale.

High-Impact WRCode Use Cases

These examples illustrate how **WRCode** and **WRConnect** can be used to deliver powerful, privacy-preserving automation across real-world settings. **WRConnect** is only mentioned when an external IoT device is actively connected.

1. Hospital Triage – Instant, Touchless Intake

Scenario: A visitor scans a **WRCode** at the hospital entrance.

- The on-device AI template asks: "*What brings you in today?*"
- Vault provides non-sensitive preferences (language, allergies) or optional PII (health ID, symptoms).
- Geofencing confirms entrance zone; no login or ID input required.
- Forms are prefilled on the user's device.
- Optimization layer can offer deeper support if needed.

WRConnect could optionally be used in this scenario — for instance, to connect the user's phone to the smartwatch for sharing health data with the connected service app from the hospital, or to interact with a hospital service robot if such infrastructure is available. **WRCode** is increadible flexible. This example is only meant to demonstrate the potential of **WRCode**.

2. 🚎 Transit Hubs – Barrier-Free Route Planning

Scenario: A traveler scans a WRCode posted at the entrance of a bus or train station. Immediately, a local AI assistant asks: *"Where are you going?"* The user answers—*"Platform 7"* or *"Bus to the airport."*

How it works:

- Scan = anchor point. The WRCode identifies the station and the exact location of the user within it (e.g., "Main entrance, ground level").
- Local map download. A signed indoor map and routing graph are loaded onto the device.
- Geofence confirmation. Each WRCode carries a geofence token that ensures the scan originates within the station. This verification is performed fully locally — no user location data is ever tracked or transmitted.
- Orientation without GPS. Instead of satellite signals, the system displays photos of the current junction with clear direction pointers. The user simply taps the view that matches their perspective, ensuring accurate "forward" orientation.
- Optional enhancements. Beacons (BLE/UWB) or GPS, if present, can provide smoother updates — but they act only as dumb transmitters, never as receivers. The app processes signals locally

to refine navigation. Because the software is Source-available, there are no hidden backdoors and tracking is technically impossible.

Why it's beneficial:

- Barrier-free travel: elderly passengers, wheelchair users, or people with vision/hearing needs receive routes tailored to their preferences stored in the Vault (e.g., step-free, max walking distance, visual or audio guidance).
- Tourist-friendly: no app installation, no registration, no GPS lock required — just scan and go.
- Low operator cost: a handful of WRCode stickers plus an open map manifest replace expensive station apps. Optional beacons can improve comfort, but they are not required and cannot be misused for surveillance.
- Privacy by design: routing and orientation happen entirely on the device. Geofencing only verifies the scan context, never the user's movements.

Impact: Transit providers lower digital barriers while improving inclusivity and passenger flow. Travelers gain stress-free navigation even in unfamiliar stations.

And the same principle is not limited to transit hubs: WRCode-based, privacy-preserving indoor navigation can be applied anywhere people need guidance — in hospitals, airports, retail stores, shopping centers, parking facilities, or any other complex environment.

A single **WRCode** can serve all passengers with tailored, real-time routing — making public transport more accessible, efficient, and privacy-preserving by design.

3. ⚡ EV Charging – App-Free Vault-Based Payment

Scenario: A driver scans a **WRCode** on an EV charger.

- Vault supplies **WRPay** token and charging preferences.
- Geofence confirms charger presence.
- **WRConnect** activates to link charger ↔ smartphone ↔ vehicle.

Impact: For the user, a single **WRCode** scan unlocks an entire charging experience — tailored, seamless, and secure. All relevant information about the charging session (duration, power consumed, cost) is collected and displayed without the need to download an app or enter credentials. If the user's vault contains a **WRPay** token and digital invoice preferences, the system can even generate a tax-compliant receipt and store it locally for bookkeeping. If the user's vault includes

preferences for taxation services (e.g., Lexware Office or similar), the **WRCode** can even trigger a follow-up **WRCode**. This secondary **WRCode** would contain a tokenized link with AI instructions tailored for automatic expense tracking, tax filing, and budget planning workflows — fully aligned with the user's accounting environment, yet without exposing sensitive data. Car and smartphone data remain connected only during the session and are not exposed to the provider. Only the absolute necessary payment details are shared.

For charging providers, this approach eliminates the need for physical cards or proprietary apps while dramatically improving onboarding. It enables anonymous but fully contextualized interactions, reduces support costs, and increases service availability — even for temporary users like tourists or rental drivers. The experience is simple to use, yet technically powerful and secure by design. And with **WRCollect** they can even offer incentives to use the service again.

4. Smart Mirror – Private In-Store Personalization

Scenario: A shopper scans a **WRCode** near a smart mirror.

A smart mirror is an intelligent, interactive display installed in retail environments — typically in fitting rooms or near product displays. It recognizes when a user is nearby and, through a secure **WRCode** scan, connects to their device to deliver personalized, real-time suggestions. These might include clothing combinations, sizing recommendations, product availability, or loyalty-based promotions.

- AI template locally triggers a styling assistant tailored to the specific store.
- Vault provides style preferences, purchase history, and favorite brands — all processed locally and anonymously.
- Geofencing ensures the interaction is location-bound and session-specific.
- The mirror can surface targeted promotions based on past buying behavior — for example, offering a discount on matching shoes if the user previously bought a jacket from the same brand.
- **WRConnect** links the user's device with the smart mirror display temporarily, enabling a fully personalized experience without exposing any identity or login data.

This setup allows retailers to enhance customer engagement with personalized service while upholding complete user privacy.

5. University Lab – Secure Device-Free Access

Scenario: A student scans a WRCode at the lab door.

- Vault provides access token.

- Geofencing ensures physical location.
- WRConnect connects user's phone to lab infrastructure for entry.

Impact: Access to lab equipment and infrastructure can be highly individualized. With **WRCode**, users don't need to log in manually — their session context, permissions, and access level can be automatically pulled from their vault and applied securely in real time. For instance, a student might have access only to specific lab equipment, while a supervisor may have full administrative control.

This process is tamper-proof and verifiable: all access actions can be logged and optionally anchored for audit purposes. Device access can be finely individualized — for example, one user may be authorized for certain lab machines while another has full administrative rights. Upon scanning the **WRCode**, the user's permissions are automatically applied from the vault without the need to log in. Device-specific settings (such as software environments, calibration presets, or tool restrictions) are loaded instantly and securely using zero-knowledge proof access tokens. This ensures accurate, compliant, and secure usage — even in critical or high-risk environments. While the provider may already know who the authorized employees are, the use of zero-knowledge proof access tokens makes sure that access is granted only to individuals with fully verifiable, permissioned rights. This adds an immutable, tamper-proof layer of trust and accountability to every interaction.



6. Live Events – Context-Aware Stream Access

A visitor scans a **WRCode** at a concert or on the website before visit. Geofencing links the wrcode to the session when arrived at the event. No need to scan the **WRCode** again..

Everyone knows the typical concert scene — thousands of people holding their phones in the air, capturing shaky, obstructed video. While this creates a personal memory, **WRCode** offers a more powerful alternative.

- After scanning the **WRCode**, the user's smartphone or AR glasses can establish a session that tailors the experience using vault preferences (e.g., favorite angles, subtitle preference, accessibility overlays). These preferences could even be preconfigured if the event organizer provides a website or ticketing portal that issues a setup **WRCode** in advance. That **WRCode** would allow the user to store desired viewing angles, notification preferences, or visual overlays in their vault.

As a bonus, **WRCollect** items like a free drink or early access pass could be gifted — embedded into the experience as part of the user's personalized event journey.

- Geofence verifies the user's location within the venue.
- **WRConnect** links the user's device to the event's public HD camera infrastructure.

This setup allows users to receive a clean, high-quality stream of the performance, even from multiple angles. In more advanced scenarios, the system can blend personal recordings with event footage, producing a professionally composed and fully personalized video summary.

Impact: Users enjoy a richer memory without recording themselves. Organizers reduce crowd distraction while offering value-added digital content. And because **WRCode** supports fully anonymous sessions, no personal data is required to enable this feature — only context and consent.

7. Voting Stations: Anonymous Civic Guidance

WRCode at each polling station provides location-specific ballot information:

- Geofence confirms district
- Displays matching voter info and explanations
- Fully anonymous — no account, no tracking

Impact: Increases trust in democratic processes, helps multilingual and first-time voters, and scales across thousands of stations. Context and geolocation, combined with minimal user input, can provide the voter with transparent, tamper-proof guidance without revealing voter identity.

8. Construction Sites: Zone-Aware Safety Briefings

Construction sites involve dynamic zones with different risks and rules. A shared **WRCode**, combined with geofencing:

- Loads local safety protocols
- Displays evacuation routes and required equipment
- Verifies WRCode and AI templates, which may include executable logic embedded directly in the QR Code, by checking them online for authenticity and compliance. All templates are anchored via cryptographic tokens on Solana and must adhere to strict publicly defined guidelines. This makes sure that safety workflows are tamper-proof, transparent, and auditable — delivering high-confidence automation without compromising user trust.

Impact: Reduces accidents, improves compliance, and simplifies onboarding for rotating crews.

Conclusion: Geofencing adds precision and context to **WRCode** automation. Together, they unlock infrastructure-grade, privacy-first workflows that are adaptive, scalable, and inclusive. From safer construction sites to smarter transportation, **WRCode + geofence** isn't just innovative — it's urgently practical. And this is only a glimpse: **WRCode** is incredibly flexible and can power a broad range of use cases, from public service delivery to secure access flows — all while preserving user privacy and auditability.

WRCode Meets OpenGiraffe: Turning Scanned Codes into Strategic AI Workflows



WRCode Meets WR CODE: Turning Scanned Codes into Strategic AI Workflows

QR codes have long been used to link users to websites, apps, or actions—but in a world where automation and AI are rapidly reshaping business and daily life, this old paradigm falls short. The future isn't about static links. It's about *actionable intelligence*. And that's where WR Codes and WR CODE intersect.

From Scan to Strategy

WR Codes (Workflow-Ready Codes) look like standard QR Codes, but they carry something far more powerful: embedded instructions for AI agents. These structured templates—cryptographically signed and transparently registered on wrcode.org following the WRCode guidelines—are what make secure, tamper-proof processing possible. Registration is not just a formality; it's the foundation that enables trusted execution, feature access, and seamless integration within the ecosystem. While customer-facing templates are fully public, auditable, and pre-scanned by AI, a small number intended for private or internal use may be withheld from public disclosure. In such cases, the publisher must clearly communicate this, and the user is shown a warning prior to execution—ensuring informed consent and accountability. Importantly, all templates—whether public or private—are registered immutably and cannot be altered post-registration. This enables the integrity of the automation instructions even in non-disclosed scenarios, maintaining trust and verifiability at all times.—transform the humble scan into a trusted automation trigger. But the true

magic begins when these WR Codes are captured and analyzed by WR CODE, the browser-based AI orchestrator developed by Optimando.ai.

Why WR CODE Unlocks WRCODE's Full Potential

When a WR Code is scanned on a mobile device, its payload is stored securely. But instead of triggering blind execution, the payload can later be routed to a local WR CODE instance—running on a workstation or even a laptop. Here, the real transformation begins:

- AI templates inside the WR Code define pre-auditable** workflows**—for example:
 - Automated bookkeeping
 - Contract generation
 - Document filing or email drafting
- These templates are passed to modular AI agents inside WR CODE, each running independently in browser tabs or a unified orchestrator page.
- The result is a fully explainable, interactive, and user-controlled automation, rather than a silent script or server-side logic.

Strategic Optimization: Where AI Meets User Intent

WR CODE doesn't just "run the AI instructions"—it reasons about them. Its built-in optimization layer evaluates potential risks, checks the user's calendar and context, and surfaces smarter or more efficient alternatives. This ensures that workflows are not only executed safely, but also aligned with the user's intent, availability, and priorities—*before* anything happens.

Imagine scanning a WR Code for a new supplier registration. Instead of blindly filling out a form, WR CODE might:

- Detect missing tax info or payment terms
- Compare past supplier entries for best practices
- Prompt you to pre-approve email drafts, contract clauses, or calendar entries

All of this happens in a transparent, modular interface, where every action can be traced, paused, adjusted—or denied.

Trust, Privacy, and Local Control

In contrast to cloud-based automation tools, WR CODE is locally hosted, privacy-first, and fully under user control. It integrates with tools like n8n for backend tasks and uses local or hybrid LLMs (e.g., GPT-4, Claude, Mistral, or LLaMA) to handle reasoning.

Each WR Code is:

- Anchored on Solana for tamper-proof authenticity and immutable verification
- Issued by verified providers who must register and prove their identity—this identity proof is also anchored on Solana
- Restricted to registered templates and context only—unregistered WR Codes are automatically rejected by apps that comply with the security guidelines defined by wrcode.org
- Verified locally before execution, ensuring the user stays in full control at all times

This layered architecture ensures that no automation can be executed unless it adheres to strict security protocols defined by wrcode.org—including user consent, verifiable audit trails, and cryptographic proof of authenticity. Every WR Code must originate from a verified provider, and only registered templates are accepted. This registration requirement makes every process tamper-proof, auditable, and enforceable, ensuring users remain fully informed and in control—whether reviewing a calendar draft, an email, or a generated report. Moreover, the secure vault used to store and execute these automations holds personally identifiable information (PII) and sensitive data. It must comply with strict implementation guidelines, and its codebase is vetted, Source-available, and cryptographically protected. Configurations are trustless—no provider can alter the code once deployed. Any modified vault implementation simply won't function within the WRCode.org

ecosystem due to real-time integrity checks. These checks ensure that only unaltered, verified, and cryptographically sealed vault configurations are permitted to operate—preserving the system's trustless and tamper-proof essence. The vault is responsible for storing personally identifiable information (PII) and sensitive data, making its integrity critical. On mobile devices, lightweight LLMs may be used to process logic securely. On workstations, the LLM configuration is flexible but always bound by tamper-proof, verifiable constraints—ensuring data sovereignty and trustworthy automation execution.

WRCode as the Standard for Secure, AI-Driven Automation

WR Codes are not proprietary—they're a proposed open standard built on QR Code model 40 (ISO/IEC 18004). Registering a WR Code is what enables secure, tamper-proof automation to unfold within the orchestration ecosystem. Whether the template is public or private, registration ensures integrity, prevents unauthorized changes, and unlocks safe, explainable automation through WR CODE. Their innovation lies in *how* they're used:

- As portable AI instruction carriers
- As context-aware automation triggers
- As secure gateways to orchestration—not black boxes

When combined with WR CODE, WR Codes become part of a next-generation infrastructure for decentralized, explainable, and privacy-preserving AI workflows.

Mobile vs. Workstation: Scale Matters

While WR Codes are typically scanned on mobile devices, the optimization layer is inherently limited by the small screen and lightweight resources available. To bridge this, WR Code scans are automatically stored on the mobile device and can be synced with the **WR CODE** orchestrator—so they can later be explored in full depth on a workstation. This unlocks advanced orchestration and reveals strategic insights that go far beyond what mobile devices alone can deliver. On mobile, a single background process handles lightweight optimization tasks if enabled, keeping the experience smooth and user-friendly. But the full power of WR Code automation unfolds on multi-screen desktop workstations running **WR CODE** or on other compatible multi-agent ai orchestrator solutions.

Here, users can:

- Visualize multiple agent outputs in parallel
- Run deep strategic analysis and cross-checks
- Trigger complex, multi-step workflows
- Leverage immersive interfaces with flexible slot-based layouts

This architecture offers unlimited flexibility and insight, enabling professionals to orchestrate intelligent, explainable, and highly customizable workflows that simply aren't possible on mobile-only systems.

Real-World Impact

Here are just a few examples:

- Accounting: Scan a WR Code on an invoice → WR CODE launches a bookkeeping automation that extracts data, cross-references contract terms, and suggests ledger entries. On supported platforms, websites can dynamically generate personalized WR Codes on the invoice itself—adapting embedded AI templates to match the user's preferences, historical settings, or account profile. The level of personalization and automation depends on how finely the service provider defines the AI instructions inside the WR Code template. When designed with care, this enables fully automated, user-tailored bookkeeping workflows with minimal effort—streamlined, secure, and fully explainable.
- Legal: A contract's WR Code triggers clause verification, AI-guided summary, and a secure signature interface.
- Field Ops: A technician scans a WR Code on-site → the orchestrator opens a live support workflow, suggests steps, and logs actions locally or immutably.

Each action is precise, auditable, and completely under human oversight.

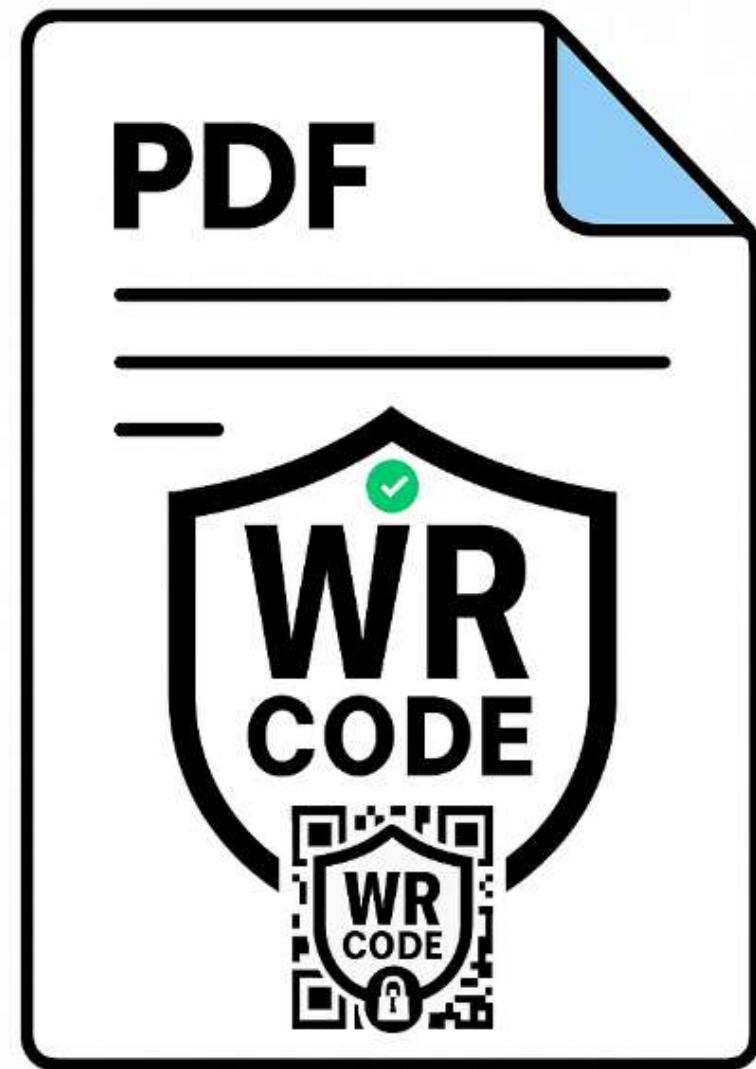
In Summary

WR Codes are the ignition key. **WR CODE** is the engine. WR Codes are already powerful when scanned on mobile devices—but they become truly magical when connected to **WR CODE**. This seamless handoff unlocks their full orchestration potential, enabling deep analysis, personalized automation, and trustworthy execution across complex workflows.

Together, they transform passive QR scans into intelligent, contextual, and strategic automation—running on systems you control, with logic you can inspect, and outcomes you can trust.

This is the future of automation—conceptual, but already taking shape. WR Codes and **WR CODE** outline a powerful vision for intelligent, secure, and explainable workflows, designed to operate in an open, local, and user-controlled ecosystem. While the full system is still evolving, the foundation has been laid—and its potential is within reach.

WRCode Stamped PDF



WRGuard

A Capsule-Based Architecture for High-Assurance Email and Document Handling

Abstract

This paper presents WRGuard, a capsule-based architecture for verifiable email and document handling, and its integration into the WRGuard Module of the WR Code Orchestrator. The system is designed to support cryptographic integrity verification, provenance attribution, and controlled interaction with email content and attachments under explicitly bounded trust assumptions.

WRGuard associates original emails, attachments, and derived representations with cryptographically verifiable metadata using WRCode identifiers, optionally anchored on public blockchains such as Solana or Optimism. Rather than treating emails or documents as safe-to-render artifacts, the architecture confines trust to Capsules: self-contained, verifiable containers that bind original inputs to deterministic parsing results and optional rasterized representations.

Parsing may be performed by WR Code Orchestrators operating with or without hardware attestation. Rasterization, however, is restricted to hardware-attested WR Code Orchestrators, either at the sender side or in explicitly isolated external environments. Sandboxed internal environments may be used for parsing in lower-risk or convenience deployments, but are not intended for enterprise or regulated environments. Within protected networks, only verified Capsule content is

consumed. Rendering engines, document interpreters, and live attachments are excluded from the trusted computing base.

This paper describes architectural design and trust semantics. It does not claim absolute security guarantees; instead, it defines a system structure intended to align with high-assurance and regulated deployment requirements through explicit trust boundaries and controlled degradation paths.

1. Introduction and Motivation

Email remains a critical communication channel for regulated industries, enterprises, and public institutions. At the same time, it continues to serve as a primary vector for phishing, spoofing, document manipulation, and unintended automation triggers. While transport encryption and digital signatures address certain aspects of authenticity, they do not sufficiently address risks introduced by:

- complex and heterogeneous rendering pipelines (HTML, PDF, fonts, images),
- discrepancies between human-visible content and machine-interpreted data,
- implicit trust in attachments and embedded objects,
- early automation on insufficiently verified inputs.

Many existing systems implicitly combine integrity verification, rendering, and interaction into a single trust domain. WRGuard and WRGuard deliberately reject this model. Instead, they introduce Capsules as an explicit intermediate representation, separating verification, transformation, rendering, and automation into distinct and auditable stages.

2. Architectural Design Principles

The WRGuard architecture is guided by the following principles:

1. Capsule-First Consumption

Raw emails, MIME bodies, and binary attachments are not consumed directly inside protected environments.

2. Envelopes remain unencrypted to preserve transparency and auditability. Parsing is performed on both sender and receiver sides: sender-provided parsing results are treated as claims, while the receiver independently parses and validates the Capsule and Envelope contents to ensure semantic fidelity. Hardware attestation is required only for trust-elevating transformations, such as Capsule encryption (when applied) and rasterization of original attachments, ensuring that protected environments admit only verifiably produced high-assurance artifacts.

3. Cryptographic verification establishes integrity and provenance across defined transformation steps. Semantic interpretation and rendering are deliberately excluded from the trust boundary.

4. Explicit Degradation Paths

Different operational environments are supported through clearly defined security levels that reduce functionality rather than expand trust assumptions.

5. Minimal Trusted Computing Base

Rendering engines, scripting runtimes, and document interpreters are excluded from the secure network by design.

3. WRGuard Core Concept

WRGuard associates emails and documents with verifiable metadata via WRCode identifiers, referencing:

- cryptographic hashes of original inputs (email body, attachments),
- hashes of derived representations (canonical text, parsing outputs, raster images),
- timestamps and generation context,
- account-bound attribution metadata.

These references may be aggregated into a commitment structure, such as a Merkle tree, and optionally anchored on a public blockchain. Integrity verification can be performed offline; anchor validation requires network access.

WRGuard itself is format-agnostic and does not prescribe rendering or interaction behavior.

4. Capsules as the Primary Trust Container

A Capsule is a self-contained, cryptographically bound container that represents an email and its associated artifacts. A Capsule may include:

- a canonical plain-text representation of the email body,
- deterministic parsing outputs for attachments,
- optional rasterized representations of the email body and attachments,
- cryptographic hashes for all included components,
- metadata describing transformation modes, parameters, and toolchain identifiers.

Within protected networks, Capsules are the only objects intended for direct consumption.

5. Sender-Side WR-Stamped Email Workflow

5.1 Composition and Stamping

In the WR-stamped workflow, Capsule construction begins at the sender side:

- The sender authenticates using a WRCode account.

- The email is authored as plain text or using registered WR templates.
 - Attachments and email bodies are included as separate encrypted original binaries, enabling transport-agnostic handling beyond the original email transport layer. Encrypted files are always treated as hostile by default and are never consumed within WRGuard-protected environments. Only encrypted Capsules originating from hardware-attested senders may be decrypted and processed inside WRGuard-protected environments, subject to explicit policy admission.
- ### 5.2 Attested Parsing and Optional Rasterization

Before transmission:

- Parsing is performed at the sender side on the original content and does not require hardware attestation, while optional rasterization is performed exclusively by a hardware-attested WR Code Orchestrator.
- Parsing produces deterministic, canonical text representations.
- Rasterization, when enabled by policy, produces static bitmap representations (e.g., WebP) that preserve visual layout.
- Original binaries are hashed and bound but are not required to be rendered downstream.

5.3 Capsule Assembly

The Capsule binds together:

- hashes of the original email body and attachments,
- hashes of canonical text and parsing outputs,
- hashes of rasterized representations, if present,
- metadata describing parsing and rasterization parameters.

All components are included in a single cryptographic commitment. The Capsule is transmitted as the primary payload.

5.4 Receiver-Side Consumption

WR-stamped Capsules may be rendered directly within the secure network on the receiver side. If the sender-side WR Code Orchestrator operated under hardware attestation, the Capsule may include embedded rasterized representations of the email body and attachments. WR-stamped emails generated without sender-side hardware attestation remain directly consumable but include only canonical text and parsing results; rasterized visual representations are omitted in this case.

6. Handling of Unstamped Emails

6.1 Level 0 — Low-Risk / Convenience Environments

In low-risk environments, unstamped emails may be capsule-converted locally:

- The host system triggers an isolated virtual machine.
- Parsing is performed within the VM on the original content.
- No rasterization is performed in this environment.
- The resulting Capsule is returned to the host environment.

The VM is invoked via a controlled handoff mechanism (e.g., one-time token and local broker), avoiding clipboard sharing or QR-based transfer. This mode is explicitly user-selected and is not intended for enterprise or regulated environments.

6.2 Level 1+ — Regulated and High-Assurance Environments

In regulated or high-assurance environments:

- Unstamped emails are not parsed or rasterized inside the secure network.
- Emails are forwarded or referenced to an external, SIM-isolated device (tablet or laptop).
- This device:
 - operates outside the protected network,

- runs a hardware-attested WR Code Orchestrator,
- performs parsing and optional rasterization,
- assembles a Capsule with cryptographic proof of derivation.

The resulting Capsule is delivered to a dedicated inbound address (e.g., wr@...) and processed identically to sender-stamped Capsules.

7. Link Handling and External Devices

7.1 Link Masking

All hyperlinks, regardless of stamping status, are masked within the secure network. Links are represented only as QR codes or WRCode references.

7.2 External Link Resolution and Re-Capsulation

Links are resolved exclusively on external, SIM-isolated devices. Retrieved content is parsed, optionally rasterized, and encapsulated exclusively by a hardware-attested WR Code Orchestrator, and then forwarded back as a sanitized Capsule.

8. Capsule Verification and Embedded Raster Artifacts

8.1 Canonical Text and Parsing Proofs

For each Capsule:

- original inputs are hashed,
- canonical text and parsing outputs are hashed independently,
- all hashes are included in the Capsule commitment.

Verification establishes integrity and provenance across declared transformation steps.

8.2 Embedded Rasterized Representations

When rasterization is enabled:

- rasterized representations (e.g., WebP) are embedded directly in the Capsule,
- each raster artifact includes a hash, page or section identifier, and encoding parameters,
- raster hashes are included in the Capsule commitment.

Recipients verify the Capsule commitment prior to extracting or decoding any raster data. Rasterized representations are treated as passive visual references and do not influence automation or policy decisions.

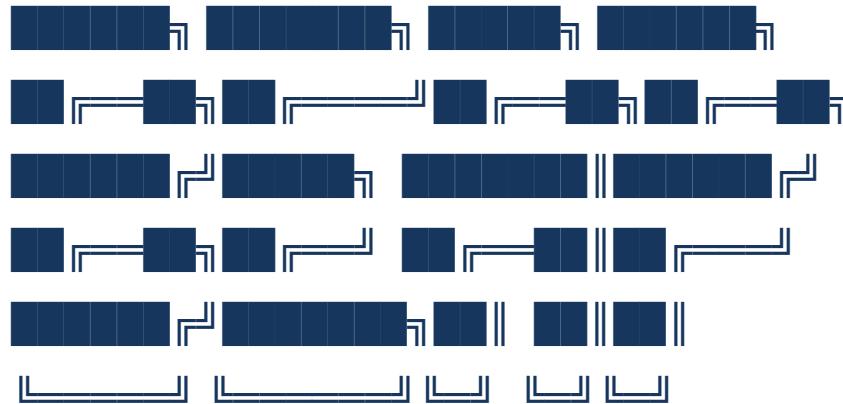
9. Display Model Inside the Secure Network

9.1 Default View

- Canonical plain-text representation.

Optional Text-Based Branding

Instead of graphical logos, users may optionally include a text-based or ASCII-style representation of their brand (e.g., "BEAP") in a designated placeholder. Such representations are treated strictly as plain text and rendered using the system's own fonts and layout rules, without importing external images, fonts, or formatting. This allows limited visual expression while preserving the system's non-rendering and zero-execution security model.



- Locally styled for readability.

- No external fonts, images, or layout logic.

9.2 Optional Visual Reference

- Static rasterized view, if present.
- Preserves original layout for human inspection.
- No interaction, scripting, or data extraction.

10. Automation and Interaction

Automation mechanisms operate only on verified Capsule data:

- unverified Capsules remain non-interactive,
- automation logic consumes structured text or template data only,
- high-impact actions require explicit policy configuration and user confirmation.

Rasterized representations are excluded from automation input by design.

11. Security Characteristics

The WRGuard and WRGuard architecture establishes:

- verifiable integrity across original and derived representations,

- explicit provenance attribution,
- auditable separation of parsing, rasterization, rendering, and automation,
- reduced exposure to untrusted rendering pipelines.

These properties align the system with high-assurance and regulated operational environments without relying on assumptions of document safety.

12. Applications

- invoice and payment processing
- regulatory and compliance correspondence
- controlled document review
- attribution of generated content
- policy-governed automation workflows

13. Limitations and Scope

The architecture does not attempt to solve semantic correctness, exploit detection, or unrestricted rendering safety. Environments requiring full document interaction must rely on separate, isolated systems. This scope limitation is intentional and reflects a conservative trust model.

Conclusion

WRGuard defines a capsule-based approach to email and document handling that prioritizes verifiable transformations over implicit trust. By confining parsing to the sender side on original content and restricting rasterization to hardware-attested orchestrators, while embedding all verified representations directly into Capsules, the system enables controlled consumption of email content within secure networks while maintaining strict trust boundaries.

Rather than expanding assumptions, the architecture constrains them—reducing the trusted computing base and aligning operational behavior with the requirements of regulated and high-assurance environments.

Envelope-Bound Capability Contracts for Deterministic and Consent-Preserving Agent Automation

Abstract

As autonomous and semi-autonomous agents increasingly perform actions with real-world consequences, existing authorization and policy mechanisms fail to provide strong guarantees of user consent, predictability, and verifiability. In particular, runtime policy evaluation, transport-layer security, and post-hoc auditing do not ensure that critical actions—such as payments, external

system mutations, or irreversible user interface submissions—were explicitly authorized prior to execution.

This paper introduces an envelope-bound capability contract model in which all critical automation capabilities are cryptographically declared and evaluated at the envelope level before any task processing occurs. The envelope functions as an authoritative execution boundary: any critical action not explicitly declared is considered invalid, even if requested or inferred within the task capsule, and results in fail-closed execution abortion. Capsules are constructed by a constraint-aware builder that derives permissible task structures directly from envelope-declared capabilities.

By separating authorization from task semantics and binding consent to immutable envelope constraints, the model enables deterministic execution, upfront consent, and verifiable enforcement even in partially compromised or adversarial environments. Optional integration with WRCode-registered counterparties further strengthens protection in fraudulent scenarios by enabling counterparty identity verification and cryptographic intent commitment.

1. Introduction

Agent-based automation systems increasingly combine reasoning, planning, and interaction with external systems. Such agents may initiate payments, modify enterprise records, trigger workflows,

or perform user interface actions traditionally reserved for human operators. While these capabilities improve efficiency, they introduce significant risks when agent behavior deviates from user intent or authorization boundaries.

Most existing systems rely on one or more of the following mechanisms:

- Transport- or channel-level security assumptions
- Runtime policy engines embedded within execution environments
- Retrospective auditing and logging

These approaches suffer from structural limitations. Runtime policies can be bypassed or misinterpreted, transport security constrains delivery but not execution semantics, and post-hoc auditing does not prevent harm. Most critically, they fail to ensure that user consent is bound to execution in a precise, enforceable, and verifiable manner *before* automation begins.

This paper proposes a model in which critical automation capabilities are declared, bounded, and cryptographically enforced at the envelope level, independent of task content, agent reasoning, or runtime environment trustworthiness.

2. Architectural Model

2.1 Envelope and Capsule Separation

The system distinguishes between two primary artifacts:

Envelope

A cryptographically signed and integrity-protected structure that binds identity, key material, and execution capabilities. The envelope determines whether and under which conditions a task may be processed.

Capsule (Task Payload)

A structured representation of the requested operation, including data, parameters, and workflow instructions. Capsules are evaluated only after envelope verification succeeds.

The envelope is authoritative. Capsule content cannot extend, override, or reinterpret envelope-declared capabilities.

2.2 Fail-Closed Execution Semantics

Envelope evaluation precedes all task processing. If envelope verification fails, execution does not begin. If a capability violation is detected at any point, execution is immediately aborted without partial completion. No downgrade, fallback, or best-effort execution is permitted for envelope-bound constraints.

3. Envelope-Bound Capability Contracts

3.1 Rationale

Traditional authorization systems encode permissions as mutable runtime state. In contrast, envelope-bound capability contracts are:

- Explicitly consented to prior to execution
- Cryptographically verifiable
- Immutable for the lifetime of execution
- Independent of agent reasoning or optimization

This shifts authorization from behavioral inference to structural enforcement.

3.2 Critical Automation

Critical automation is defined as any action that produces irreversible, externally observable, or liability-relevant effects, including:

- Monetary transactions or financial commitments
- Data egress across trust boundaries

- Modification of external systems or records
- User interface actions that trigger submissions or confirmations
- Any action whose misuse may cause material, legal, or security impact

Critical automation is treated as a first-class capability class requiring explicit declaration.

3.3 Mandatory Declaration Rule

All critical automation capabilities must be explicitly declared in the envelope.

If a capsule requests, implies, or derives a critical action not declared in the envelope, execution is immediately aborted.

Formally:

A capsule may request actions, but only actions whose capability class is declared in the envelope are executable. Any undeclared critical capability results in fail-closed termination.

No implicit authorization, heuristic escalation, or runtime approval is permitted.

4. Envelope-Bound Monetary Authorization

4.1 Motivation

Allowing agents to initiate payments introduces irreversible side effects. In adversarial or partially compromised environments, runtime checks and UI-level validation are insufficient. Once a payment is executed, remediation is often impossible.

For this reason, monetary actions require pre-consented, immutable authorization constraints evaluated before execution begins.

4.2 Envelope Constraints

An envelope may declare a monetary authorization capability with immutable constraints, including:

- Maximum authorized amount
- Currency or asset identifier
- Payee identity allowlist (hashed or referenced)
- Permitted payment mechanisms
- Validity window and non-replay token
- Required authentication strength

The envelope does not encode payment instructions, only the bounds within which payment actions may occur.

4.3 Enforcement Properties

Any payment attempt outside declared bounds is rejected. Successful execution yields verifiable proof that the action conformed to pre-consented constraints. This proof is independent of agent reasoning and runtime environment trust.

4.4 WRCode-Registered Counterparties in Fraudulent Scenarios

Envelope-bound egress control constrains *where* a payment action may be executed, preventing unauthorized redirects or endpoint substitution. However, egress control alone does not constrain *how* an authorized endpoint interprets or commits to the action once reached.

In fraudulent or adversarial scenarios, WRCode-registered sites or software provide an additional security layer by enabling counterparty identity verification and cryptographic intent commitment. A registered site can be required to authenticate using a stable, registry-anchored identity and to produce signed commitments over critical execution parameters such as amount, currency, and recipient identifiers.

This mechanism mitigates attack classes that remain possible under strict egress control, including semantic manipulation at the server level, undisclosed reassignment of payment recipients, or

divergent interpretation of submitted parameters within opaque backend systems. By requiring that the counterparty explicitly attests to the same intent declared in the envelope, execution becomes bound not only to an authorized path but also to an authorized interpretation.

WRCode registration does not replace envelope-level enforcement. Instead, it augments envelope-bound capability contracts by transforming external counterparties from assumed-honest endpoints into cryptographically accountable participants, strengthening execution guarantees in high-assurance and fraud-sensitive environments.

5. Envelope-Bound Deterministic Action Manifests

5.1 Motivation

UI and interaction automation is traditionally specified through low-level event sequences that are brittle and difficult to verify.

5.2 Action Manifest

The envelope may declare an Action Manifest constraining the permissible interaction space, including:

- Permitted target identities or origins

- Allowed action types
- Maximum number of externally visible actions
- Deterministic preconditions and postconditions
- Explicitly forbidden transitions

The manifest constrains what may happen, not how the agent reasons. In scenarios involving third-party payment processors, envelope-declared constraints are applied within the egress boundaries of the execution environment and used for consent verification, while the processor remains a constrained execution endpoint without semantic awareness of the envelope.

5.3 Deterministic Enforcement

If runtime conditions prevent the manifest from being satisfied—due to UI changes, unexpected redirects, or external interference—execution fails closed rather than adapting unpredictably.

6. Capability Mismatch Detection

6.1 Runtime Classification

During execution, attempted actions are classified by capability class based on observable effects rather than agent intent.

6.2 Abort Semantics

If an attempted action belongs to a critical capability class not declared in the envelope, execution is immediately aborted. No partial side effects are permitted.

This prevents:

- Implicit privilege escalation
- Prompt or payload injection
- Policy drift due to agent optimization
- Accidental overreach

7. Constraint-Aware Capsule Construction

7.1 Capsule Builder Role

Capsules are constructed by a constraint-aware builder that:

- Reads envelope-declared capabilities
- Restricts capsule structure and semantics accordingly
- Prevents generation of capsules requiring undeclared critical automation

7.2 Architectural Consequences

This design ensures:

- Structural safety by construction
- Early detection of authorization mismatches
- Stable envelopes with dynamic but bounded capsules

The builder does not grant permissions; it ensures alignment with pre-consented constraints.

8. Security and Threat Analysis

8.1 Addressed Threats

- Policy bypass via agent logic manipulation
- Unauthorized side effects in compromised environments
- Replay or reuse of prior authorizations
- Undeclared automation triggered through inference

8.2 Residual Risks

- External system changes may cause fail-closed aborts
- Envelope complexity must be managed to avoid excessive regeneration

These are intentional trade-offs favoring determinism and verifiability over permissiveness.

9. Discussion: Consent as an Execution Boundary

By requiring that all critical automation be declared in the envelope and aborting execution on any undeclared capability, consent becomes an enforceable execution boundary rather than a user-interface artifact.

The envelope functions as a cryptographic capability contract, while the capsule represents a bounded request operating strictly within that contract. This separation enables third-party verification that no undeclared critical actions occurred, without access to task content or agent reasoning.

10. Conclusion

Envelope-bound capability contracts provide a structural mechanism for enforcing consent, predictability, and determinism in agent-based automation systems. By elevating authorization to a

cryptographically verifiable envelope layer and mandating explicit declaration of critical automation, the model reduces reliance on trusted runtimes and mitigates common failure modes of dynamic policy systems. Optional integration with WRCode-registered counterparties may further strengthen protection in fraudulent or adversarial scenarios by extending verifiable intent across organizational boundaries.

In such integrations, WRCode-registered sites or software providers are associated with a registry-backed cryptographic identity, allowing critical actions—particularly in high-stake transaction contexts—to be consistently attributed to a specific counterparty. This identity association supports post-execution verification and auditability by enabling correlation between pre-consented envelope constraints and externally executed actions, without requiring assumptions about the internal correctness or trustworthiness of the counterparty's runtime environment.

Policy-Bound Egress Control in BEAP Capsules

Deterministic and Time-Spaced Limitation of Automation Egress via Cryptographic Envelope Binding

Abstract

Automation workflows increasingly operate across heterogeneous infrastructures and administrative domains, where execution environments cannot be assumed to be uniformly trusted or controlled. In

such settings, outbound behavior ("egress") represents a critical risk surface, as automation logic may trigger network communication, data forwarding, or secondary actions beyond its intended scope.

The Bidirectional Email Automation Protocol (BEAP) introduces a policy-bound approach to egress control by embedding egress constraints directly into the cryptographic envelope of a capsule. These constraints are cryptographically bound to capsule identity and evaluated by policy-aware orchestrators in combination with locally enforced policies. Effective egress is restricted to deterministic, explicitly whitelisted paths and is applied in a time-scoped manner during execution, making deviations observable at the protocol boundary.

1. Motivation

In most automation frameworks, egress control is treated as an operational concern, enforced through configuration files, adapter logic, network firewalls, or API gateways. Such mechanisms depend heavily on the correctness of the runtime environment and often allow dynamic resolution of destinations and side effects during execution.

In practice, automation frequently involves third-party adapters, delegated execution environments, or partially trusted orchestration layers. Under these conditions, unintended egress behavior—such as redirection, repackaging, or uncontrolled forwarding—can occur without violating local runtime assumptions.

BEAP addresses this by treating egress as a protocol-level property. Egress intent is declared upfront, cryptographically bound to the capsule, and evaluated deterministically during execution rather than inferred at runtime.

2. Architectural Context

2.1 Capsules and Cryptographic Envelopes

A BEAP capsule consists of structured automation data enclosed within a cryptographic envelope.

The envelope binds together:

- Capsule identity
- Integrity metadata
- Policy descriptors
- Egress constraint descriptors

This binding ensures that declared automation intent, including its permissible outbound effects, remains inseparable from the capsule's identity and integrity representation.

2.2 Coordinated Enforcement and Policy Hierarchy

Execution in BEAP is intentionally not bound to a single control layer. Egress control is designed as a coordinated mechanism spanning the orchestrator, sandboxed execution environments, and strict operating-system-level enforcement.

Policy-aware orchestrators validate the capsule envelope prior to unpackaging and automation and execute automation logic inside sandboxed containers or similarly isolated execution contexts. These execution contexts are subject to explicit operating-system-level egress rules that constrain network access and external communication.

The capsule policy expresses the maximum permissible scope of automation behavior as defined by the sender. Local orchestrator policy is authoritative and always takes precedence. Effective permissions are derived from the intersection of capsule policy and local policy, with local policy able to further restrict, delay, or deny execution regardless of capsule intent.

Egress is permitted only if:

1. the capsule policy allows the class of outbound action, and
2. the local orchestrator policy explicitly whitelists a compatible egress path.

This hierarchy ensures that capsules cannot expand local permissions and that outbound behavior remains under administrative control of the execution environment.

3. Policy-Bound Egress Constraints

3.1 Definition of Egress

Within BEAP, egress refers to any action that produces externally observable effects, including:

- Network communication
- Emission or forwarding of capsules
- Interaction with external systems
- Transfer of data beyond the controlled execution boundary

3.2 Cryptographic Binding of Egress Constraints

Egress constraints are embedded into the capsule envelope as structured policy elements and included in the cryptographic material used to establish capsule integrity.

Conceptually, the envelope binds:

`EnvelopeBinding = Hash(`

`CapsuleContent ||`

`CapsulePolicyDescriptor ||`

```
EgressConstraintDescriptor
```

```
)
```

Any modification to declared egress constraints results in a different binding value and can be detected during envelope verification. This ensures that egress intent cannot be altered without breaking the cryptographic linkage between capsule content and policy.

4. Deterministic and Time-Spaced Egress Resolution

Egress resolution in BEAP is deterministic and explicitly time-scoped to the execution phase of an automation. Orchestrators do not dynamically discover destinations or infer permissions at runtime. Instead, outbound actions are resolved against a finite set of locally whitelisted egress paths.

Resolution proceeds as follows:

1. The envelope is verified and capsule policy is extracted.
2. Local orchestrator policy is applied and intersected with capsule policy.
3. Automation logic is executed inside a sandboxed container with OS-level egress restrictions applied.
4. Outbound actions are matched against canonical, locally whitelisted egress identifiers.

During active execution, only explicitly permitted egress paths are enabled. All other outbound channels are expected to be blocked for the duration of execution. If an outbound action cannot be mapped unambiguously to a locally whitelisted egress path compatible with the capsule policy, execution is declined, halted, or requires explicit operator consent and local whitelisting.

Determinism is achieved through explicit whitelisting and temporal scoping rather than implicit trust in adapters, endpoints, or runtime context.

5. Execution in Adverse Environments

BEAP is designed for execution scenarios in which orchestration environments may be misconfigured, partially compromised, or operated by third parties. In such cases, policy-bound and time-scoped egress constraints reduce the range of outbound effects that can occur without policy conflict and increase the visibility of deviations at the protocol level.

While this approach does not guarantee correct behavior under all circumstances, it establishes a structured and verifiable mechanism for constraining egress behavior independent of transport security and runtime assumptions.

6. Threat-Oriented Analysis

6.1 Considered Threats

| Scenario | Description |
|------------------------------|--|
| Modified orchestration logic | Attempts to expand or alter outbound behavior |
| Malicious or faulty adapters | Redirect or repurpose automation outputs |
| Endpoint substitution | Runtime redirection to unintended destinations |
| Capsule replay | Reuse of capsules outside intended execution scope |

6.2 Design Implications

Cryptographic binding of egress constraints makes undetected modification of outbound permissions more difficult. Local policy supremacy ensures that execution environments retain full control over allowed egress paths. Sandboxed execution combined with OS-level enforcement supports time-scoped restriction of outbound channels, while deterministic resolution reduces ambiguity and reliance on mutable runtime state.

Residual risks remain where orchestrator implementations are non-compliant or intentionally malicious, but deviations become structurally observable at the policy and verification layer.

7. Conclusion

Policy-bound egress control in BEAP elevates outbound automation behavior from a runtime configuration concern to a protocol-level property. By embedding egress constraints into the cryptographic envelope of a capsule and enforcing them through locally authoritative orchestrators, sandboxed execution environments, and OS-level rules, BEAP enables deterministic and time-scoped control over automation egress paths across heterogeneous environments.

Rasterization and Reconstructable Office Documents in Capsule-Based Systems

Overview

Enterprise communication extends beyond PDFs and routinely involves editable office document formats such as word processing files, spreadsheets, and presentations. The Capsule architecture generalizes the PDF rasterization approach to these formats in order to provide a uniform, format-agnostic handling model that separates semantic content from visual layout while preserving the ability to deliver editable documents to the receiver.

Commonly Used Enterprise Document Formats

The model applies equally to the most widely used business formats, including:

- Word processing: DOCX, DOC, ODT, RTF
- Spreadsheets: XLSX, XLS, ODS, CSV
- Presentations: PPTX, PPT, ODP
- Text-based formats: TXT, Markdown, HTML

Ingestion and Capsule Representation

For all supported formats, document ingestion follows the same principle:

1. Text Parsing

The document's textual content is extracted and normalized into the Capsule as plain text or typed fields. This representation becomes the canonical semantic reference used for automation, validation, and later reconstruction.

2. Visual Rasterization

The full document layout is rasterized into static image representations (e.g., WebP). These images capture pagination, layout, tables, charts, and embedded graphics and serve as a visual reference equivalent to PDF previews.

The Capsule therefore contains both a structured textual representation and a faithful visual snapshot of the original document.

Receiver-Side Document Reconstruction

Unlike PDFs, many enterprise documents are expected to remain editable after delivery. To support this, the receiver does not obtain the original file but a reconstructed document derived from Capsule data:

- Format-Specific Reconstruction Models

Specially trained, format-aware LLMs rebuild a near-original editable document (e.g., DOCX, XLSX, PPTX) using the rasterized images as layout guidance.

- Text Reconciliation

Reconstructed text is aligned with and replaced by the exact textual content stored in the Capsule, ensuring semantic equivalence.

- Visual Element Recovery

Images, tables, charts, and layout elements are extracted or reconstituted from the rasterized representation and embedded into the reconstructed file.

The result is an editable document that closely matches the original in appearance and structure while remaining fully derived from the Capsule representation.

Summary

By extending rasterization beyond PDFs to all major office document formats and combining it with format-specific reconstruction, the Capsule architecture enables safe previews, deterministic semantic handling, and delivery of editable, near-original documents—without relying on direct reuse of the original file.

Note on Macros and Active Document Logic (Out of Scope)

In theory, active document logic such as macros (e.g., VBA modules in DOCM/XLSM/PPTM files) could be extracted as textual source code, encapsulated as a separate Capsule payload, and re-embedded or reconstructed on the receiver side as part of a rebuilt editable document. Such an approach would conceptually align with the separation of semantic content and presentation by treating macros as an explicit, non-visual payload class.

However, this approach is outside the scope of BEAP and is not recommended. Macros represent executable logic with complex environmental dependencies, non-deterministic behavior, and context-sensitive side effects that are not reliably captured through rasterization or canonical text extraction. Reconstructing or re-injecting such logic would exceed the design goals of BEAP, which focus on deterministic content exchange, verifiable provenance, and controlled automation.

Accordingly, BEAP treats macros and comparable active document logic as theoretically transportable but intentionally unsupported. Any macro-related handling is considered experimental, policy-dependent, and explicitly excluded from standard BEAP-compliant document exchange.

Deterministic Reconstruction and Trustless Verification in BEAP

The reconstruction of PDF documents and email bodies from rasterized representations is conceptually feasible in the same manner as for other commonly used office and enterprise document formats, including word processing files, spreadsheets, and presentation documents. In this model, rasterized artifacts serve exclusively as non-executable visual references, while semantic content is represented separately as normalized text or structured instructions. On the receiver side, this information may be deterministically reassembled into a near-original representation under strictly controlled conditions, without rendering or executing the original source files.

A fundamental design principle of the BEAP Capsule is the transformation of potentially harmful, ambiguous, or insecure content into controlled representations. These representations consist either of plain-text instructions suitable for deterministic processing or of rasterized artifacts that are explicitly non-executable. Any reconstruction performed on the receiver side follows deterministic rules and does not rely on embedded logic, scripts, macros, or other executable constructs originating from the original content.

Within the BEAP protocol, rasterization, parsing, integrity checks, and identity checks are security-critical operations and are required to be performed on-premise. This requirement ensures that all trust-relevant transformations can be independently inspected, reproduced, and verified without

reliance on opaque third-party infrastructure. By enforcing local execution under verifiable hardware attestation, the system enables external security experts to deterministically evaluate the trustless system architecture used for secure enterprise communication.

Rasterization produces non-executable visual artifacts, while identity verification binds those artifacts to a specific, attestable toolchain and configuration, ensuring that the visual representations are provably derived from the declared inputs and processes. Integrity checks ensure that parsing outputs, normalized text representations, and derived artifacts remain consistent and untampered throughout the Capsule lifecycle. Together, these mechanisms prevent substitution attacks, non-deterministic transformations, and hidden execution paths that could undermine trust.

For this reason, rasterization components, parsing components, integrity-check mechanisms, and identity-check mechanisms within BEAP are limited to source-available original implementations, subject to explicit license permissions. The required verification properties depend on the ability to audit, reproduce, and attest the exact transformation logic. Cloud-based execution environments are excluded by default from BEAP-compliant processing.

Cloud integration is permitted only for Capsule Builders and only if they operate using BEAP-provided, downloadable and locally executable modules. These modules MUST enable on-premise execution of rasterization, parsing, integrity verification, identity checks, and hardware-backed attestation under trust assumptions equivalent to fully local deployments.

Orchestrators that receive BEAP Capsules and perform orchestration or capability execution MUST be source-available or open source. This requirement ensures that external security experts can audit and verify that orchestration logic strictly defaults to explicit user consent prior to any policy enforcement or capability activation. Under no circumstances may Capsule reception alone trigger execution or policy application.

The overarching objective of BEAP is to establish a reproducible, verifiable standard for secure enterprise automation and communication. OpenGiraffe is defining a protocol with strict, reproducible guidelines to ensure that BEAP Capsules are secure by default, independent of the generating system, deployment model, or vendor.

By design, this architecture minimizes implicit trust in document formats, rendering engines, and execution environments, while enabling interoperable, high-assurance workflows across heterogeneous enterprise infrastructures.

Handling of Attachments and Time-Based Media in Capsules

Attachments and time-based media such as documents, binaries, audio, and video are treated under a strict trust-boundary model.

Any attachment that could contain executable logic, active content, or non-deterministic structures is never rendered or executed within the Capsule trust boundary.

Capsules therefore do not embed active or directly consumable attachments.

Original files are always treated as external artifacts and are not opened inside the trusted network environment.

Encryption and Policy-Controlled Decryption

Potentially harmful original attachments MUST be encrypted and cryptographically bound to the Capsule context.

Decryption is only permitted outside the trusted network boundary and requires an explicit WR Code or QR Code scan, subject to the receiver's local security policy.

The Capsule itself does not perform, trigger, or imply decryption, execution, or rendering of original attachments.

All decisions regarding decryption, access, or denial are enforced by the receiving environment.

For enterprise deployments, this separation is strictly enforced.

Policy rules MAY prohibit decryption entirely, restrict it to isolated environments, or require additional authorization steps before external access is granted.

Semantic Representation of Attachments

Instead of embedding original files, Capsules MAY include deterministic, non-executable representations of attachments, such as:

- parsed and schema-validated text content,
- rasterized previews or snapshots,
- immutable metadata describing the attachment,
- cryptographic hashes (fingerprints) of the encrypted originals.

These representations enable automation, inspection, and auditability without exposing the system to execution or decoding risks.

Overlay-Based External Access

When encrypted original attachments are referenced, an explicit overlay MAY be displayed that includes:

- relevant rasterized previews (if applicable), and
- a WR Code or QR Code used to initiate access outside the trusted network boundary.

This overlay ensures that access to original attachments is intentional, user-initiated, and policy-controlled, and that trust-boundary transitions are clearly visible to the user.

Enterprise Enforcement Model

In enterprise environments, the following rules apply:

- Original attachments remain encrypted by default.
- Decryption inside the trusted network boundary is prohibited.
- External access is allowed only if explicitly permitted by policy.
- Automated workflows operate solely on deterministic representations, never on original files.

This enforcement model prevents implicit execution, reduces attack surface, and preserves auditability and compliance requirements.

Security Rationale

By enforcing encryption of original attachments, restricting decryption to external, policy-controlled environments, and operating internally only on deterministic representations, the system avoids:

- hidden execution paths,
- decoder and parser vulnerabilities,
- implicit trust escalation,

- nondeterministic automation behavior.

This design keeps Capsules secure by default, verifiable, and suitable for high-assurance and enterprise-grade automation.

Normative Security Requirements (BEAP Capsules)

BEAP Capsules MUST prohibit polymorphic type resolution and class-bound deserialization.

Capsule payloads MUST be parsed into neutral, side-effect-free data structures and validated against strict schemas. Payload data MUST be treated as inert and declarative at all times.

Execution behavior MUST NOT be inferred from payload structure, field names, nesting, or type annotations.

Execution behavior MUST be determined exclusively by explicitly allowed capabilities configured in the receiving orchestrator.

Capsule payloads MUST NOT contain executable logic, implicit control flow, dynamic references, or implementation-specific type identifiers.

Canonicalized Capsule payloads MUST be cryptographically hashed and verified against their associated integrity metadata prior to validation, capability resolution, or any downstream processing. Capsules failing verification MUST be rejected and MUST NOT be processed further.

Threat Model (BEAP Capsule Processing)

1. Scope and Assumptions

The BEAP threat model assumes a hostile and adversarial content environment. Capsule payloads MUST be treated as untrusted, regardless of their origin, transport channel, or apparent intent.

The model assumes that:

- Payloads may be malformed, adversarially crafted, or intentionally misleading.
- Payload authorship and payload structure MUST NOT be trusted.
- Attackers may attempt to exploit parsing behavior, deserialization mechanisms, or implicit execution semantics.

The threat model focuses on protocol-level facts and does not rely on trust in implementations, operators, or payload sources.

2. Threat Classes and Mitigations

2.1 Code Execution via Deserialization

Threat:

Payloads attempt to trigger code execution by abusing:

- polymorphic type hints,
- embedded class identifiers,
- reflection-based instantiation,
- constructor or initializer side effects.

Mitigation (Mandatory):

- Polymorphic type resolution MUST be disabled.
- Class-bound deserialization MUST NOT be used.
- Payloads MUST be parsed into neutral value trees only.
- Parsing MUST NOT invoke constructors, hooks, or runtime logic.

2.2 Implicit Execution Semantics

Threat:

Payload structure, naming conventions, or nesting patterns implicitly influence execution paths or behavior.

Mitigation (Mandatory):

- Payloads MUST be strictly declarative.
- Execution behavior MUST NOT be inferred from payload structure or content.
- All executable behavior MUST be gated behind explicitly allowed capabilities.
- Capability resolution MUST occur exclusively within the receiving orchestrator's static policy domain.

2.3 Parser Ambiguity and Gadget Chains

Threat:

Payloads exploit parser ambiguities such as:

- duplicate keys,
- tolerant or non-deterministic parsing,
- non-canonical encodings,
- non-standard extensions.

Mitigation (Mandatory):

- Deterministic and strict JSON parsing MUST be enforced.

- Duplicate keys, non-canonical encodings, comments, and extensions MUST be rejected.
- Payloads MUST be canonicalized prior to hashing.
- Canonicalized payloads MUST be cryptographically hashed and verified before any further processing.
- Any verification failure MUST result in immediate Capsule rejection.

2.4 Confused Deputy and Privilege Escalation

Threat:

Payloads cause execution under elevated privileges by implying authority, intent, or trustworthiness.

Mitigation (Mandatory):

- Capsule payloads MUST carry no authority.
- Capabilities MUST be resolved exclusively by the receiving orchestrator.
- Capability execution MUST be constrained by local policy and configuration, not by payload content.

2.5 Injection via Context Switching

Threat:

Payload values are embedded into execution contexts such as:

- SQL or NoSQL queries,
- HTML or template engines,
- shell commands or scripts.

Mitigation (Mandatory):

- Payload parsing MUST be strictly separated from execution contexts.
- Context-specific escaping and validation MUST be applied at execution boundaries.
- Capsules themselves MUST remain context-agnostic and inert.

3. Explicitly Out-of-Scope Threats

The BEAP protocol does not claim to eliminate:

- implementation bugs in orchestrators or adapters,
- misconfigured capability policies,
- vulnerabilities in external systems invoked by explicitly allowed capabilities.

Such issues are considered implementation or operational risks, not protocol-level failures.

4. Core Security Invariant

The fundamental BEAP security invariant is:

At no point MAY Capsule payload data directly or indirectly determine executable code paths, instantiate runtime objects, or influence execution semantics beyond explicitly permitted capabilities.

A Capsule is considered valid only if:

- its payload is canonicalized,
- its hash is verified,
- its schema validation succeeds,
- and capability execution is explicitly authorized by the receiving orchestrator.

Capsules failing any of these conditions MUST be rejected.

1. RFC-Style MUST / MUST NOT Table (BEAP Capsules)

BEAP Capsule Processing – Normative Requirements

ID Requirement

- R-1 Capsule payloads MUST be treated as untrusted input at all times.
- R-2 Capsule payloads MUST NOT include executable code, scripts, macros, or dynamic references.
- R-3 Polymorphic type resolution MUST NOT be used during payload parsing or deserialization.
- R-4 Class-bound deserialization MUST NOT be used.
- R-5 Payloads MUST be parsed into neutral, side-effect-free data structures (e.g. value trees).
- R-6 Payload parsing MUST NOT invoke constructors, hooks, initializers, or runtime logic.
- R-7 Payloads MUST be validated against strict, predefined schemas.
- R-8 Execution behavior MUST NOT be inferred from payload structure, field names, or type annotations.
- R-9 Execution behavior MUST be determined exclusively by explicitly allowed capabilities in the receiving orchestrator.

ID Requirement

- R- 10 Deterministic and strict JSON parsing MUST be enforced.
- R- 11 Duplicate keys, non-canonical encodings, comments, and extensions MUST be rejected.
- R- 12 Capsule payloads MUST be canonicalized prior to integrity verification.
- R- 13 Canonicalized payloads MUST be cryptographically hashed and verified before any further processing.
- R- 14 Capsules failing integrity verification MUST be rejected and MUST NOT be processed further.
- R- 15 Payload parsing MUST be strictly separated from execution contexts (SQL, HTML, shell, templates).
- R- 16 Capability execution MUST be constrained by local orchestrator policy, not payload content.

ID Requirement

R-
17 Capsule encryption and rasterization are restricted to hardware-attested senders

2. Formal BEAP Capsule Processing Pipeline (Step 0 – N)

This pipeline is normative. Any deviation violates BEAP compliance.

Step 0 – Capsule Intake

- Capsule is received from any source.
- No trust is assumed in origin, transport, or sender identity.

Step 1 – Strict Parsing

- Payload is parsed using deterministic, strict JSON rules.
- Parsing fails if:
 - duplicate keys are present,
 - non-canonical encodings are detected,
 - non-standard extensions are used.

→ No execution, no binding, no interpretation.

Step 2 – Canonicalization

- Parsed payload is converted into a canonical JSON representation.
- Canonicalization allows comparison of identical logical content → identical byte representation.

Step 3 – Integrity Verification

- Canonicalized payload is cryptographically hashed.
- Hash is verified against Capsule integrity metadata (e.g. WRStamp).

→ Failure → immediate rejection. No fallback.

Step 4 – Schema Validation

- Verified payload is validated against strict BEAP schemas.
- Unknown fields, invalid types, or out-of-bounds values are rejected.

Step 5 – Neutral Representation

- Payload is materialized as a neutral value tree.
- No class instantiation.

- No reflection.
- No runtime logic.

Step 6 – Capability Resolution

- Orchestrator evaluates:
 - whether referenced capabilities exist,
 - whether they are permitted under local policy.

→ Payload does not influence this decision.

Step 7 – Controlled Execution (Optional)

- Only explicitly permitted capabilities may be executed.
- Execution occurs outside the payload domain.
- Payload values are treated as data inputs only.

Step 8 – Audit & Proof (Optional but Recommended)

- Hashes, decisions, and capability invocations may be logged.
- Enables deterministic replay and third-party audit.

Security Claims and Limitations (BEAP)

BEAP defines a capsule-based data exchange model designed to minimize execution risk arising from untrusted structured content. The protocol enforces strict separation between declarative payload data and executable behavior.

Under correct and compliant implementation, BEAP provides the following security properties:

- Capsule payloads are treated as inert data and cannot directly or indirectly trigger code execution.
- Execution behavior is exclusively controlled by the receiving orchestrator through explicitly permitted capabilities.
- Canonicalization and cryptographic verification provide tamper evidence and deterministic validation of payload integrity.
- Strict parsing and schema validation reduce ambiguity and parser-level exploitation risks.

BEAP does not claim to eliminate:

- vulnerabilities in orchestrator or adapter implementations,

- misconfigured capability policies,
- security flaws in external systems invoked by allowed capabilities.

These risks are considered operational or implementation-specific and are outside the protocol's formal structure.

BEAP is designed to be auditable, deterministic, and suitable for high-assurance environments, provided that implementations adhere strictly to the normative requirements defined in this specification. In BEAP, data may express intent, but it can never select, instantiate, or execute code. Orchestrators that adopt the BEAP standard MUST adhere to the same strict protocol guidelines and MUST use only approved BEAP modules for parsing, verification, and validation of Capsule content. Custom or substituted parsing, verification, or integrity mechanisms are not permitted unless they are functionally equivalent, auditable, and explicitly approved under the BEAP specification.

This requirement ensures that Capsule handling remains deterministic, reproducible, and externally verifiable across all compliant orchestrator implementations, and prevents divergence that could reintroduce implicit trust or execution risk. BEAP aims to establish a secure, trustless, transport-agnostic standard for enterprise communication and automation.

Deterministic Automation, Policies, and Threat Analysis in BEAP

BEAP is designed to enable secure, trustless, and auditable enterprise automation while maintaining strict control over execution authority. To achieve this, BEAP enforces a clear separation between verified communication channels, authoritative policies, and execution logic.

Handshakes vs. Policies

In BEAP, a handshake and a policy serve distinct and non-overlapping roles.

A handshake establishes a verified communication channel between two parties, such as a user and a company, or between two organizations. It provides cryptographic assurance of identity, channel integrity, provenance, and—where applicable—hardware or system attestation. A successful handshake enables secure communication only. It does not grant execution authority and cannot enable automation by itself.

A policy, by contrast, is an authoritative configuration defined independently by each party. Policies determine whether automation is permitted, under which conditions it may occur, and which capabilities may be executed with or without interactive consent.

Sender Policies and Receiver Policies

BEAP explicitly distinguishes between sender-side request policies and receiver-side authoritative policies.

The sender may attach a request policy that expresses intended actions, constraints, or preferred automation parameters. This request policy is advisory in nature and serves to communicate intent, expectations, or optional constraints.

The receiver maintains its own authoritative policy. This policy evaluates incoming requests and deterministically allows, restricts, escalates, or rejects automation based on local preferences, security standards, compliance requirements, and risk tolerance.

Execution authority is derived exclusively from the receiver's policy. Sender-defined request policies cannot grant execution privileges, bypass controls, or override receiver-side decisions.

Policy-Authorized Automation

Receiver-side policies may explicitly allow certain classes of actions or capabilities to execute automatically without per-request user consent. Such automation is always:

- explicitly configured,
- deterministic,
- auditable,
- and scoped to predefined conditions.

Automation in BEAP is therefore never implicit. The existence of a handshake or a request policy alone is insufficient to permit execution.

Deterministic Threat Analysis as a Safety Mechanism

Even when automation is explicitly authorized by receiver policy, BEAP allows the application of local threat analysis as an additional safety mechanism. This analysis does not alter Capsule semantics and does not introduce implicit behavior. Instead, it operates as a policy-governed evaluation step.

Threat analysis may combine multiple local techniques, including:

- OCR applied to rasterized content to identify potentially harmful, deceptive, or unexpected material,
- deterministic parsing of structured Capsule data,
- local machine learning and fine-tuned LLMs to identify anomalies, deviations from expected patterns, or suspicious capability combinations.

All analysis is performed locally, using auditable tools, without cloud-based execution or external data transfer.

Policy-Controlled Outcomes

Threat analysis outputs—such as anomaly indicators or risk scores—are treated as inputs to deterministic receiver-side policy logic. Policies may define that certain analysis outcomes:

- block execution,
- require escalation or additional review,
- downgrade execution privileges,
- or trigger enhanced logging and warnings.

Any such outcome MUST be explicitly defined in policy, reproducible, and auditable. Probabilistic analysis components never independently authorize execution and never override receiver authority.

Consent and Visibility

Where receiver policies require interactive consent, threat analysis results MAY be used to augment the consent interface, highlighting abnormalities or elevated risk at the moment of decision. Where policies allow fully automated execution, threat analysis may still trigger policy-defined blocking or escalation to prevent misuse, abuse, or unexpected behavior—even within trusted channels.

At no point can a handshake, a sender request policy, a Capsule, or a probabilistic analysis component implicitly trigger execution.

Core Design Principle

BEAP enforces a strict separation between:

- communication trust (handshakes),
- execution authority (receiver policies),
- truth-establishing mechanisms (deterministic parsing, verification, identity checks),
- and advisory analysis mechanisms (OCR, ML, LLM-based threat analysis).

This architecture enables high-assurance, policy-driven automation while preserving explicit control, determinism, auditability, and trustless execution across heterogeneous enterprise environments. In BEAP, handshakes establish verified channels, sender policies express intent, receiver policies grant execution authority, and deterministic safeguards ensure that automation is never implicit.



BEAP

Bidirectional Email Automation Protocol

BEAP: A Bidirectional Email Automation Protocol for AI-Driven Workflows

The Bidirectional Email Automation Protocol (BEAP) is a new application-layer standard designed to transform email from a passive communication channel into a secure, machine-actionable automation interface. While SMTP, JMAP, and traditional email markup systems provide transport or UI enhancements, they do not define how organizations can exchange structured automation, agent instructions, or semantic context. BEAP fills this gap by introducing a cryptographically verifiable request/response model that enables multi-AI-agent workflows, semantic embeddings, and natural-language-defined mini-apps to be transmitted directly through email.

Technical Foundation

BEAP operates as a strict, typed automation capsule embedded in standard email messages (Content-Type: application/beap+json). Capsules follow a rigid schema that accepts only predefined fields, ensuring deterministic and verifiable execution across all orchestrators.

A BEAP Capsule contains:

- Intent metadata describing the requested operation.
- Multi-AI-agent orchestration parameters defining routing, model selection, tool permissions, and agent collaboration logic.
- Embedded context payloads (typically < 5 MB for ~1000 pages), which the receiver semantically embeds into its local SQLite vectorstore, either temporarily or permanently. PostgreSQL can be configured as backend database optionally.
- Optional sensitive context or automation references, allowing data to be resolved and processed exclusively on the receiver's device—a key strength enabling privacy-preserving automation.
- Workflow IDs, Mini-App IDs, and Augmented Overlay configurations providing exact, reproducible UI states, interactive overlays, deterministic workflow paths

- AgentBoxes and orchestrator configuration blocks, enabling precise setup of agent behavior, tool bindings, runtime constraints, and environment-specific execution rules.
- rasterized email bodys and attachments from hardware attested sender environments.
- Desired response behavior, defining output formats, return codes, follow-up Capsule types, or chained workflow requirements.
- Automation policies specifying safety levels, approval steps, fallback conditions, and execution boundaries.
- Cryptographic signatures, hashes, and key identifiers ensuring authenticity, integrity, replay protection, and compatibility with WRCode/WRStamp verification.

This structure turns email into a secure, verifiable automation channel capable of orchestrating complex multi-agent workflows, embedding semantic context, rendering interactive overlays, and executing sensitive logic entirely on the local orchestrator.

The protocol follows a straightforward state machine:

receive email → verify envelope → apply policy → execute agents → send BEAP response.

Execution is handled locally by a lightweight BEAP Gateway running on customer infrastructure. The gateway interfaces with vector databases, LLM orchestration layers, and mini-app runtimes, enabling high-security offline or hybrid workflows without exposing automation logic to third parties.

Key Differentiators

1. Multi-AI-Agent Automation

BEAP is the first email-based protocol designed for orchestrating multi-agent reasoning pipelines. A BEAP request can launch extractor agents, validators, transformers, or custom operational agents, enabling fully automated B2B processes without APIs or webhooks.

2. Semantic Context Embeddings

Email becomes context-aware. BEAP capsules can contain context for local embeddings into a vector database or reference vector IDs or content hashes stored locally, enabling retrieval-augmented reasoning, memory continuity, and structured document interpretation across organizations.

3. Natural-Language-Defined Mini-Apps

Instead of embedding HTML or interactive scripts, BEAP carries declarative mini-app instructions. These are converted into local components and executed in a sandboxed environment, enabling interactive dashboards, forms, validators, and process visualizers without loading external JavaScript.

4. Cryptographic Verification

BEAP signs the intent, payload, and referenced templates at the application layer. This prevents tampering, replay attacks, and unauthorized automation. Unlike DKIM or S/MIME, BEAP signatures bind the *automation semantics*, not just the message body.

Advantages

- Universal transport: works over any existing mailserver without altering SMTP/JMAP.
- Zero API onboarding: partners only need an email address to exchange structured automation.
- High security: BEAP verifies every instruction, template, and context reference before execution.
- AI-native workflows: multi-agent orchestration, embeddings, and vector stores are first-class participants.
- Offline and on-prem capability: automation executes inside a local BEAP Gateway, independent of cloud availability.
- Incremental adoption: organizations can respond manually, semi-automatically, or fully automatically.

Mini-App Library Principle (Three-Tier Architecture)

A core part of the BEAP ecosystem is the tiered Mini-App Library, designed for extremely low latency and predictable behavior. Mini-apps, agent boxes, and atomic code blocks all share the same JSON-based schema, enabling real-time synthesis or modification.

Tier 1 — Top-Level Library (Mini-Apps)

The system first queries the high-level library of complete mini-apps.

If a similar app already exists (same layout, intent, or function), it is reused without modification.

Tier 2 — Mid-Level Components (Agent Boxes & Settings)

If no suitable full mini-app is found, the system checks whether individual components can be adapted.

Agent boxes, settings panels, and logic templates are modular and can be reused with minor adjustments.

Tier 3 — Atomic Code Blocks

If neither a mini-app nor a component fits, the system builds the required mini-app from vetted atomic code blocks stored in the lowest tier.

These blocks form the foundation for consistent, secure, and deterministic app synthesis.

Outcome: Extremely Low Latency

Because BEAP's mini-app system always reuses or modifies existing blocks before generating new ones, the orchestration layer achieves:

- very low computation times
- stable, predictable output

- high security through code provenance
- optimized local performance without heavy LLM processing

Conclusion

BEAP establishes a new paradigm: automation exchanged as messages.

By combining multi-agent reasoning, cryptographically signed intent, semantic context embeddings, a tiered mini-app and Augmented Overlay synthesis model, BEAP enables organizations to build secure, interoperable, AI-driven automation pipelines using nothing more than email.

Semantic Processing, Recommendation, and Orchestration Pipeline

To support BEAP's tiered Mini-App Library and the tiered Augmented Overlay Library, the system implements a multilayer semantic decision architecture. It determines whether an incoming request—originating from a BEAP capsule, WR-Chat, Augmented Overlay interaction, or contextual event—can be fulfilled by reusing an existing Overlay element or Mini-App (Tier 1), by adapting a component (Tier 2), or by constructing a new element from atomic blocks (Tier 3).

All core logic is implemented in TypeScript, with the option to move performance-critical modules to Rust in future builds.

1. NLP Layer: Intent Extraction for Tier Selection

The NLP subsystem provides the semantic foundation for selecting or synthesizing Mini-Apps and Augmented Overlay elements.

1.1 High-Level NLP (warp-nlp)

warp-nlp is the current high-level intent engine and provides:

- fast, rule-oriented intent detection
- trigger and pattern recognition for WR-Chat and Overlay interactions
- coarse semantic matching against existing Tier-1, Tier-2, and Tier-3 elements
- deterministic output without neural inference

This ensures immediate routing of high-level BEAP intents or user instructions to the appropriate synthesis tier.

1.2 Low-Level Embedding Models (optional)

When a request requires finer semantic discrimination—e.g., differentiating between closely related Augmented Overlay widgets or similar Mini-App templates—the system may use:

- neural embedding models via TensorFlow.js or a similar solution
- local LLM embedding pipelines

- domain-specific parsers

These generate dense representation vectors that complement warp-nlp's (or any other nlp solution) high-level intent signals.

2. Unified Feature Representation for Mini-Apps and Overlays

All textual, structural, and contextual data is converted into a unified semantic feature map used in the tier selection process:

1. Text normalization (canonicalization, lemmatization, stopword removal)
2. Vector embeddings
 - warp-nlp high-level vectors
 - optional neural embeddings
3. Metadata encoding for both Mini-Apps and Augmented Overlay elements:
 - function type (UI navigation, data entry, automation, form generation, guidance)
 - expected input/output schema
 - security classification

- interaction mode (Overlay widget, Mini-App panel, adaptive hint, routed action)

4. Execution context

- domain/page state
- WRCode handshake and security requirements
- recent interactions and historical success metrics
- device or browser constraints

Mini-Apps and Augmented Overlay elements share the same structured representation, enabling a unified decision-making pipeline.

3. Tiered Selection for Both Mini-Apps and Overlays

Both libraries follow the same hierarchical model:

Tier 1 — High-Level Elements (Complete Mini-Apps & Full Overlay Widgets)

If a full solution already exists, the orchestrator selects it immediately.

Examples:

- A full invoice-submission Mini-App

- A complete “highlight navigation path” Overlay widget
- A reusable “explain this page” overlay assistant

The system ensures extremely low latency by reusing Tier-1 elements whenever possible.

Tier 2 — Mid-Level Components (Agent Boxes, UI Blocks, Logic Panels, Overlay Sub-Widgets)

If no Tier-1 match exists, the system checks whether an element can be assembled by adapting components, e.g.:

- Agent Boxes with pre-configured logic
- Settings panels
- Reusable UI fragments
- Localized overlay components such as arrows, tooltips, contextual hints, query selectors

These components can be recombined with minimal transformation.

Tier 3 — Atomic Blocks

If Tier-1 and Tier-2 are insufficient, the system synthesizes a new Mini-App or Overlay element from vetted atomic units:

- atomic UI blocks
- atomic DOM-interaction primitives
- atomic logic fragments
- atomic state-handling routines

These blocks form the cryptographically verifiable base of BEAP automation.

4. Recommender Layer: Vector Ranking + Hard Constraints

The recommender determines which tier element should be used for the current intent.

4.1 Vector-Based Ranking

Each Mini-App, Overlay element, component, and atomic block is stored with a semantic vector signature.

The system ranks candidates based on:

- semantic similarity
- structural compatibility
- execution context

- historical usage patterns

4.2 Deterministic Constraints

Before a candidate is approved, it is validated against a deterministic rule set:

- WRCode handshake requirements
- security/privacy constraints
- schema compatibility
- domain restrictions
- allowed automation scope

This ensures that adaptive ranking cannot produce unsafe or invalid output.

5. Reinforcement-Learning Policy (Optional, Bounded by Constraints)

An optional RL policy refines selection and ordering within the allowed space.

RL does not create new actions; it only optimizes which safe Tier-1/Tier-2 recommendation is likely to satisfy a given intent.

State Representation

The RL state includes:

- success rates of Tier-1 and Tier-2 matches
- historical preference patterns
- contextual metadata
- latency and efficiency metrics

Reward Function

Rewards reflect:

- reduced need for Tier-3 synthesis
- lower latency
- stable, predictable outputs
- minimal user intervention

6. Architectural Integrity and IP-Relevant Properties

Across all BEAP automation flows—Mini-Apps, Augmented Overlay, and atomic code synthesis—the system maintains:

- a unified semantic representation for all interactive and computational elements
- a tiered, deterministic synthesis pipeline shared by Mini-Apps and Augmented Overlays
- strict constraint enforcement as a safety boundary
- optional adaptive improvement via RL without compromising determinism
- consistent system behavior regardless of implementation language
- cryptographically provable provenance through WRCode anchoring

This architecture provides a clearly defined, technically novel framework for structured, verifiable, message-driven automation, supported by a scalable library of Mini-Apps and Augmented Overlay components.

The BEAP Capsule: A Cryptographically Secure, Transport-Agnostic Channel for AI-Driven Workflows

The Bidirectional Email Automation Protocol (BEAP) introduces a universal mechanism for exchanging structured automation instructions between organizations, systems, or AI agents. Its core building block—the BEAP Capsule—is a compact, self-contained, cryptographically verifiable capsule inside the envelope that transforms existing communication channels into secure, machine-actionable automation pathways.

A Transport-Agnostic Automation Envelope

Unlike traditional automation systems that depend on APIs or controlled infrastructures, BEAP Capsules are designed to operate across any medium capable of delivering structured data:

- Email
- P2P messaging platforms (WhatsApp, Signal, Telegram Web, etc.)
- Enterprise messaging tools
- Direct file exchange
- Local IPC, P2P links, or offline QR handover
- **The receiving endpoint is not limited to desktop systems; Capsules may be processed by orchestrator software running on heterogeneous devices, including smartphones, wearables, VR/AR headsets, robots, or embedded machines.**
- **Because all integrity, identity, and policy data reside inside the Capsule itself, it remains independently verifiable regardless of how it was transmitted.**

Every Capsule contains:

- Typed metadata describing intent, payload, and execution rules
- Account identifiers for sender and recipient

- Automation policies defining whether actions require approval or may run autonomously
- Integrity anchors referencing versioned WRCode/WRTemplate definitions
- Response triggers that instruct the orchestrator on how to continue the workflow

The Capsule is digitally signed and—when required and allowed—encrypted before transmission. All validation occurs locally, independent of transport security.

Cryptographically Tamper-Proof by Design

Each BEAP Capsule is signed using a key bound to a registered WRAccount. Upon receipt, the orchestrator:

1. extracts the raw Capsule bytes
2. canonicalizes them
3. verifies the signature against the sender's public key
4. validates timestamps, replay protection, and template hashes
5. checks policy and account context

Crucially, verification occurs without rendering the Capsule.

The orchestrator processes it strictly as data, isolating it from any surrounding message content. This

protects the system even when Capsules arrive through insecure, mixed-content environments such as consumer-grade messengers or untrusted email bodies.

If a Capsule is intercepted, modified, or re-packaged inside harmful content, verification fails automatically.

Embedded Response Triggers for Fully Automated Workflows

A BEAP Capsule defines not only its content but also its lifecycle.

Response triggers embedded inside the Capsule tell the orchestrator how to react:

- generate a structured automated response
- request human approval
- execute a mini-apps, AI agents or Augmented Overlay instructions
- update semantic context or vector stores
- initiate multi-step workflows
- Because triggers are part of the Capsule itself, the orchestrator does not rely on fragile external configuration or pre-defined routing rules.

Organizations can exchange automation logic simply by exchanging Capsules.

Secure Operation Across Messaging Platforms Without Rendering

Messaging platforms—especially those with web interfaces like WhatsApp Web, Signal Desktop, Telegram Web, Slack, or Discord—are ideal transport channels because BEAP Capsules can be extracted and verified without displaying or rendering them.

The process is:

1. Raw message content is captured from the DOM or API.
2. WRGuard isolates potential Envelope and Capsule blocks from untrusted context.
3. Capsules are validated strictly as binary/text data.
4. Only verified Capsules are allowed into the orchestration environment.

This prevents:

- phishing attempts
- HTML or script-based attacks
- contextual deception
- accidental execution of unsafe attachments

- harmful preview rendering

The Capsule model ensures that only cryptographically validated automation instructions can trigger actions.

WRGuard and Augmented Overlay: Enterprise-Grade Protection for Human Interaction

While BEAP secures automation, WRGuard secures user interaction around it.

WRGuard enforces two core principles:

1. Automation Is Isolated From Human Message Content

Even if an attacker embeds a valid Capsule inside a malicious message, WRGuard:

- sanitizes the full message
- blocks unsafe attachments
- prevents rendering of harmful HTML
- isolates Capsules in a secure pipeline
- displays only a safe overlay representation for humans

Only the Capsule's verified data influences automation. The surrounding message content is treated as untrusted.

2. Safe Link Handling Through the Augmented Overlay

To prevent employees from clicking potentially dangerous links in business environments, WRGuard replaces all external URLs and attachments with a secure Augmented Overlay mechanism:

- Instead of displaying a clickable link, WRGuard shows an interactive hover indicator.
- Hovering reveals a WRCode, which can be scanned using a physically isolated tablet with its own SIM card—completely outside the corporate network.
- This ensures that verification and link resolution occur out-of-band, in a fully isolated environment.
- The company network remains protected even if the external content is malicious.

This model eliminates classic phishing vectors and reduces the attack surface to near zero.

A Unified Fabric for Automated, Trustless Inter-Organizational Workflows

With the introduction of the BEAP Capsule, email, messaging apps, P2P channels, and offline handovers all become equally capable transport layers for secure automation.

Organizations, developers, and AI systems can now exchange:

- structured automation intents
- mini-app instructions
- Augmente Overlay behaviour
- multi-agent workflows
- semantic context
- response requests

—without requiring new infrastructure or trusted intermediaries.

Receive Policy

BEAP Capsules specify how they must be handled when arriving on the recipient device.

By default, the orchestrator operates in a protective manual-approval mode for all incoming automation.

Users may override this default on a per-sender or per-relationship basis, enabling automatic or policy-based execution only for trusted parties.

Receive policies support three execution modes:

- Automatic — the Capsule is verified and executed immediately.
- Manual — the user must approve the Capsule before any processing occurs.
- Policy-based — rules inside the Capsule determine whether the message runs automatically, escalates to manual approval, or is rejected.

This ensures that sensitive or high-impact actions cannot trigger without explicit user consent unless trust has been clearly established.

Response Policy

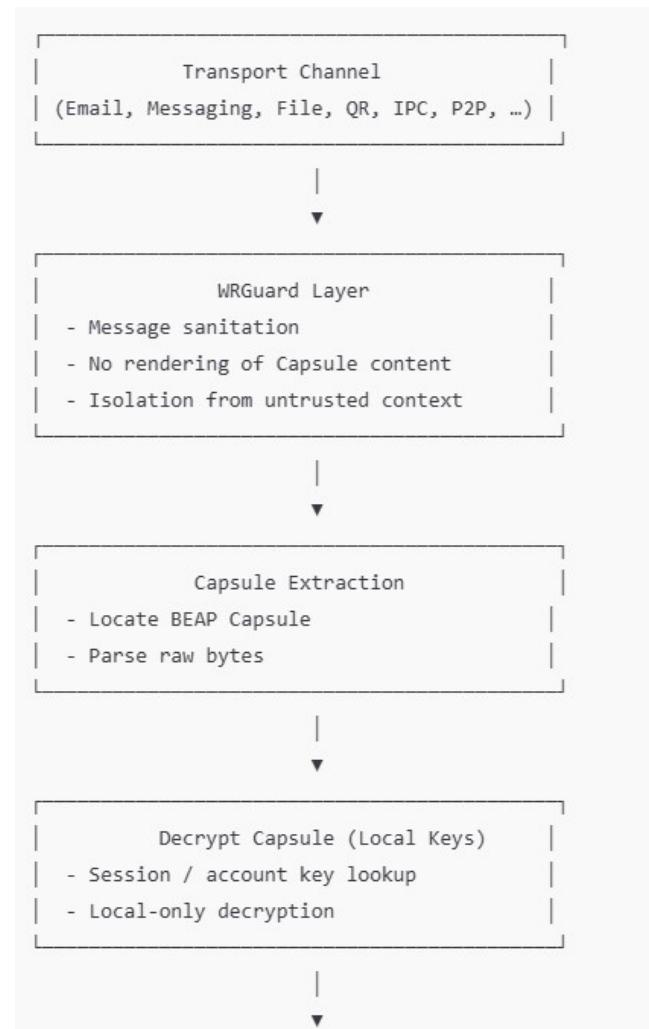
After execution, the orchestrator generates a response according to the Capsule's declared response policy. Outgoing automation also defaults to manual-approval mode, unless the user has configured trusted relationships that allow automatic replies.

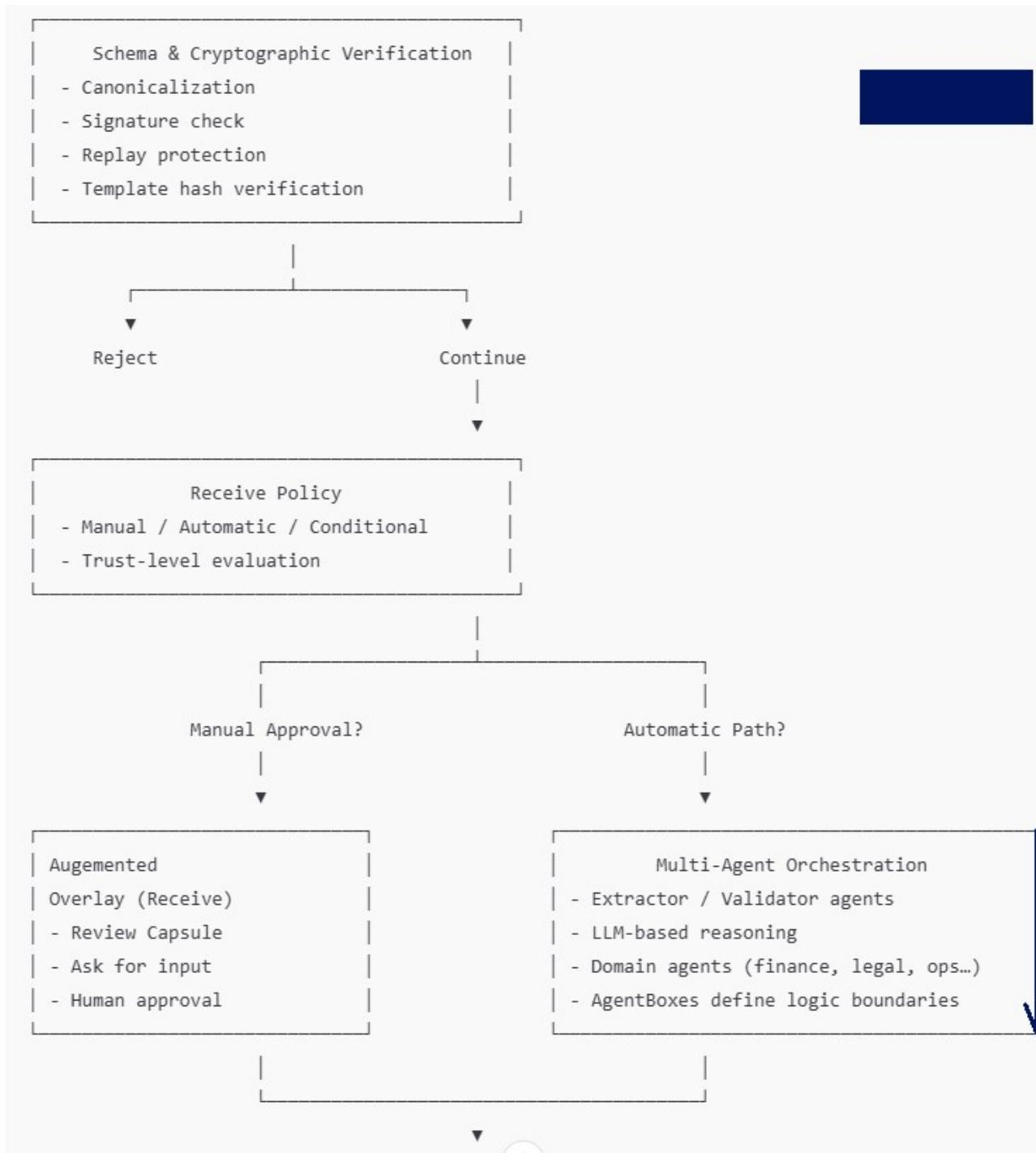
Response modes mirror the receive modes:

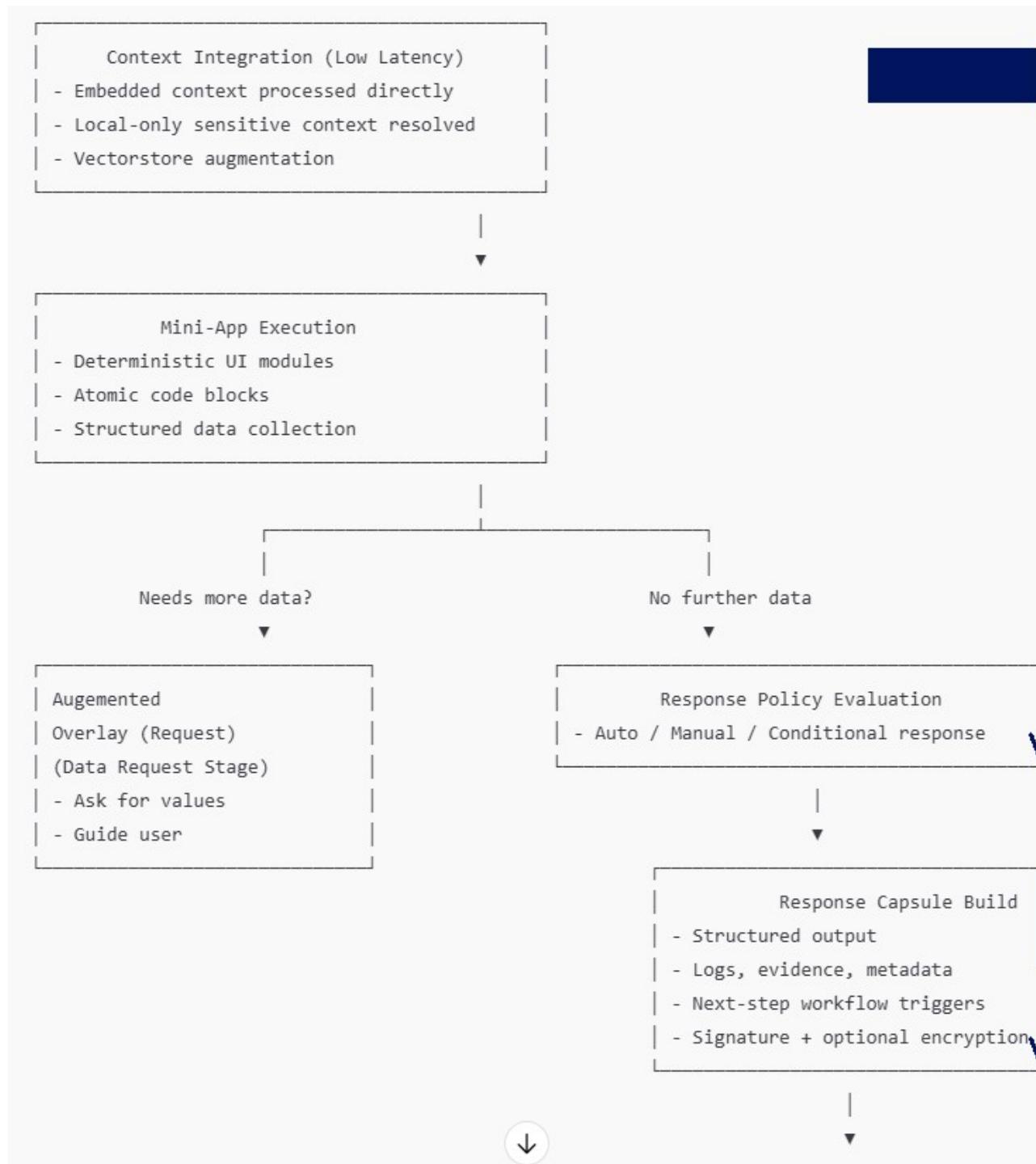
- Automatic — the orchestrator sends the response without user involvement.
- Manual — the user must approve the outgoing data.
- Policy-based — rules decide whether the response is automatic, requires approval, or must be blocked.

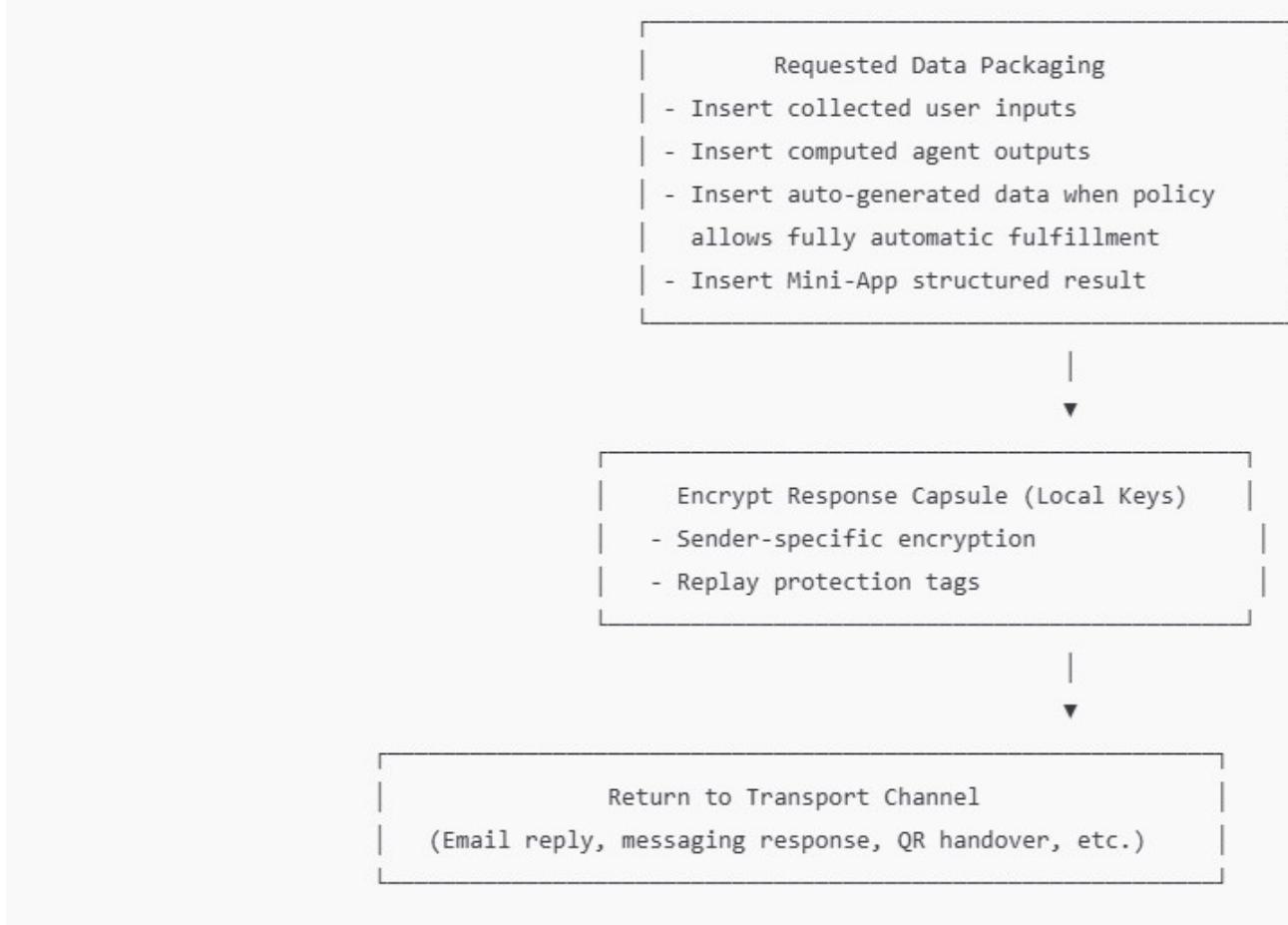
Combined receive and response policies create a clear, verifiable trust boundary for all machine-driven communication.

BEAP FLOW









Beat Flow End

The BEAP Capsule replaces the need for secondary protocols.

It is both the message and the contract: a cryptographically enforced, self-describing automation unit that remains trustworthy regardless of where it travels or how it is delivered.

The BEAP Envelope Model

Identity-First Admission and Non-Rendered Transport for Secure Email and Messaging Systems

Abstract

This article defines the BEAP (Bidirectional Email Automation Protocol) envelope model and its application across email and messaging transports. BEAP enforces a strict separation between transport and presentation by treating all protocol payloads as non-rendered containers. Message admission is performed exclusively through a minimal, plaintext identity descriptor that cryptographically binds a single, opaque BEAP container to a verified sender. Only after successful admission are system-generated, safe representations exposed to users. The model preserves compatibility with existing mail servers, spam filters, and messaging infrastructures while providing a strong trustless security architecture against spoofing, tampering, replay, and active-content attacks.

1. Transport Does Not Imply Rendering

Email and messaging systems transport byte sequences accompanied by metadata; they do not mandate how these bytes must be rendered or interpreted. Rendering is a client-side decision. Contemporary clients already separate transport from presentation by blocking inline images, suppressing unsafe attachments, or replacing encrypted payloads with status indicators.

BEAP formalizes this separation by defining all protocol artifacts as transported but non-rendered by design.

2. The BEAP Container Concept

A BEAP message is conveyed as a standards-compliant email or message containing a single, opaque BEAP container file. The container is a deterministic, versioned data structure processed only by BEAP-aware clients.

The container is never unpacked, traversed, or interpreted as a filesystem object.

3. Internal Logical Structure of the Container

While externally represented as a single file, the BEAP container logically binds:

3.1 Identity Descriptor (Plaintext, Non-Rendered)

A schema-constrained descriptor containing:

- sender identity references,
- receiver role references,
- cryptographic signatures,
- integrity hashes covering the entire container,
- protocol and schema version identifiers,
- additional signals such as enterprise user roles.

- .

This descriptor is the sole admission authority.

3.2 Encrypted Capsule (Non-Rendered, hardware-attested sender cryptographically verified)

An encrypted structure encapsulating:

- canonicalized message semantics,
- structured representations of message content,
- references to original artifacts.

3.3 Encrypted Originals (Non-Rendered)

The original message body and attachments, preserved only for archival, compliance, or controlled redistribution.

These components are conceptually distinct but cryptographically bound into a single inseparable object.

4. Identity-First Admission Model

The only entry point for a BEAP message is the identity descriptor embedded in the container.

Upon receipt, a BEAP-capable client performs, in strict order:

1. schema validation of the identity descriptor,
2. verification of cryptographic signatures against a WR Code-verified sender,
3. integrity verification by recomputing hashes over the entire container.

Successful admission proves that the complete container was sent by the verified sender and has not been modified. No parsing, decryption, or presentation is permitted prior to admission.

5. Non-Rendered Processing Semantics

All BEAP containers are explicitly non-rendered. They are never:

- displayed,
- previewed,
- executed,
- implicitly decrypted,
- or automatically opened.

Non-rendered does not imply inaccessibility. Clients MAY store, forward, archive, or export BEAP containers strictly as opaque byte sequences, but MUST NOT bind them to any renderer, execution environment, preview mechanism, or implicit processing path.

Possession of a container does not imply the ability to interpret, decrypt, or execute its contents.

5.1 Policy-Governed Opening Location and Authority

The ability to open or process a BEAP container is governed by a dual-policy model consisting of:

- a Capsule Policy, embedded within the container, and
- a Receiver Policy, defined and enforced by the receiving environment.

The effective execution policy is derived exclusively through policy intersection.

At no point may the Capsule Policy elevate, override, or weaken privileges defined by the Receiver Policy.

The Receiver Policy always has final authority.

The Capsule Policy MAY impose minimum admissibility requirements, including but not limited to:

- required orchestration class (e.g. enterprise orchestrator),

- required execution environment (e.g. hardware-attested),
- required operator role (e.g. CTO).

If the Receiver Policy cannot satisfy these requirements, processing MUST abort and fail closed.

5.2 Opening Method Resolution

If policy resolution succeeds, the opening method is determined by the effective execution policy and MAY include:

- fully automated unpacking,
- unpacking only after explicit operator consent,
- export for controlled relocation (e.g. transfer to an air-gapped environment).

The opening method is treated as a security-relevant control decision, not as a user-interface convenience.

No implicit fallback behavior is permitted.

5.3 Absence of Server-Side Profiling

The BEAP processing model records no server-side or infrastructure-level profiling or telemetry metadata, including but not limited to:

- open events,
- render attempts,
- execution signals,
- user interaction traces.

All policy evaluation and enforcement occur locally within the orchestrator-controlled environment.

As a result, confidentiality and policy enforcement remain intact even under partial or full infrastructure compromise.

Air-Gapped Capsule Generation, Transport, Finalization, and Reception

5.4 Architectural Roles and Authority Model

BEAP enforces a strict separation between intent construction, identity authority, capsule finalization, transport, and decryption authority.

Within this architecture:

- Only the online orchestrator operating under an authenticated WR Code account is identity-authoritative.
- Air-gapped orchestrators operate exclusively as subordinate (sub-orchestrators).
- Transport devices never possess semantic or identity authority.
- Decryption authority is never implicit and is governed exclusively by policy.

This separation ensures enforceable identity lifecycle control, revocation capability, and accountability, while preventing offline or transport-level entities from escalating privileges.

5.5 Air-Gapped Capsule Builder Mode (Input Exfiltration Mitigation)

BEAP supports an optional air-gapped capsule builder mode for high-assurance deployments.

This mode is not introduced due to deficiencies in capsule construction or orchestration logic. Capsule construction is deterministic, auditable, and cryptographically constrained by design.

The motivation for air-gapped generation is the risk of input-side exfiltration at the underlying system layer during intent formation, including:

- keystroke capture,

- screen or framebuffer scraping,
- compromised input peripherals,
- malware operating below the orchestrator layer.

Air-gapped builder mode relocates the moment of intent expression into an environment where such observation paths are structurally eliminated rather than merely detected.

This mode is restricted to enterprise-grade orchestrators with strong hardware attestation. Baseline BEAP interoperability does not require air-gapped operation.

5.6 Unfinalized Capsule Construction on Air-Gapped Sub-Orchestrators

Air-gapped orchestrators operate exclusively as sub-orchestrators.

They are permitted to:

- construct capsule intent and payload,
- define capsule policy constraints,
- produce a cryptographically signed unfinalized capsule.

They are explicitly not permitted to:

- bind authoritative WR Code identity,
- assert reception eligibility,
- finalize capsules for third-party reception.

An unfinalized capsule:

- is immutable with respect to intent and policy,
- is not directly receivable by external parties,
- does not assert authoritative identity.

5.6.1 Encrypted Addressing to the Main Orchestrator

All unfinalized capsules generated in air-gapped builder mode are cryptographically addressed to a specific main orchestrator.

This encrypted addressing ensures that:

- only the designated main orchestrator can securely receive the capsule,

- untrusted hosts or relay environments cannot intercept or substitute capsule data,
- confidentiality and integrity are preserved during transit.

Encrypted addressing does not imply decryption authority.

The main orchestrator may receive and finalize a capsule without accessing plaintext content. Whether, where, and by whom decryption may occur is determined exclusively by policy and evaluated only after finalization.

5.7 Sub-Orchestrator and Transfer Device Pairing

Before an air-gapped orchestrator may contribute capsules, it MUST be cryptographically paired as a verified sub-orchestrator of a specific main orchestrator.

This pairing process involves:

- the air-gapped sub-orchestrator,
- the online, identity-authoritative main orchestrator,
- one or more wallet-grade transfer devices used for constrained identity exchange.

All participating orchestrators MUST be enterprise-grade and hardware-attested.

The pairing establishes:

- cryptographic identity lineage,
- subordinate authorization scope,
- admissible capsule contribution rights,
- admissible transport device identities.

Only paired sub-orchestrators and paired transfer devices may participate in capsule workflows.

5.8 Capsule Transmission as Opaque Byte Transport

Capsule transmission is treated strictly as opaque byte forwarding.

Transmission mechanisms MAY include:

- network transport,
- removable media,
- air-gapped transfer paths.

Transmission environments are assumed to be potentially compromised. At no point does the transmission layer gain semantic authority over capsule content, policy, or execution context.

The capsule remains unchanged throughout transmission.

5.9 Wallet-Grade Transfer Devices and Port-Bound Admission

For air-gapped workflows, BEAP supports wallet-grade transfer devices, comparable in assurance level to enterprise hardware wallets or HSM-adjacent devices.

Such devices are characterized by:

- cryptographically verifiable, hardware-bound device identity,
- immutable or securely updatable firmware,
- strictly restricted USB device classes (no HID, no composite behavior),
- absence of general-purpose execution or rendering capabilities,
- deterministic, protocol-bound byte transfer only.

In air-gapped environments, the orchestrator MUST enforce port-bound admission policies:

- each designated transfer port is bound to a specific device identity,
- only explicitly permitted devices MAY be admitted,
- any deviation MUST result in immediate fail-closed termination.

This prevents the physical interface from being repurposed as a general input or execution channel.

5.10 Capsule Finalization by the Main Orchestrator

Upon receipt of an encrypted, unfinalized capsule from a verified sub-orchestrator, the main orchestrator performs capsule finalization.

Finalization includes:

1. verification of sub-orchestrator identity and attestation,
2. validation of capsule structure, intent, and policy,
3. enforcement of WR Code account state and revocation status,
4. binding of authoritative WR Code identity,
5. cryptographic sealing of the finalized capsule.

Finalization does not imply decryption.

The main orchestrator may finalize a capsule without ever accessing plaintext content. Only after finalization does the capsule become eligible for reception by external receivers, subject to policy.

5.11 Air-Gapped Relocation as a Containment Mechanism

Relocation of a BEAP capsule to an air-gapped system is a supported and intentional operation intended for secure containment, not for altering execution semantics.

The same orchestrator logic, validation rules, and policy semantics apply in both online and air-gapped environments. Relocation affects only the execution substrate, not the capsule's meaning, authority, or admissibility conditions.

Capsules—whether unfinalized or finalized—MAY be transferred via wallet-grade transport devices strictly as opaque byte sequences. The transport medium does not interpret, modify, or authorize capsule content.

Finalized capsules MAY be transported to and admitted by any enterprise-grade orchestrator capable of evaluating the embedded policies and identities.

Unfinalized capsules, by contrast, are cryptographically bound to their corresponding main orchestrator and MUST only be transported to and admitted by that orchestrator for finalization. Any attempt to admit an unfinalized capsule in a different environment MUST be rejected and fail closed.

This enables the following property:

Even if the surrounding network or host system of an initial receiver is compromised, a BEAP capsule can be forwarded unchanged to a clean, air-gapped environment where policy evaluation, admissibility checks, and controlled opening may safely occur.

No trust is placed in the compromised environment beyond its ability to forward opaque data.

5.12 Capsule Retrieval and Non-Rendered Processing

Capsule retrieval follows non-rendered processing semantics in all environments.

Capsules are never:

- displayed,
- previewed,
- executed,
- implicitly decrypted,
- or automatically opened.

Retrieval does not imply execution or decryption. Capsules remain opaque until all admission checks and policy evaluations succeed.

5.13 Policy Resolution Rule (Normative)

Let:

- P_{Cj} be the Capsule Policy
- P_{Rj} be the Receiver Policy

The effective execution policy P_{e_j} is defined as:

$$P_{e_j} = P_{Cj} \cap P_{Rj}$$

Subject to the following constraints:

1. The Capsule Policy MUST NOT grant privileges not already permitted by the Receiver Policy.
2. The Receiver Policy MAY impose stricter constraints than those requested by the Capsule Policy.
3. If the intersection is empty or unsatisfiable, processing MUST abort and fail closed.
4. Successful policy resolution is a prerequisite for any decryption, unpacking, or execution step.

This rule enforces monotonic privilege reduction and prevents policy-based privilege escalation.

5.14 Revocation of Sub-Orchestrators and Transfer Devices

The main orchestrator retains exclusive revocation authority.

At any time, the main orchestrator MAY revoke:

- a paired air-gapped sub-orchestrator,
- a wallet-grade transfer device,
- or both.

Revocation immediately invalidates:

- the ability of the sub-orchestrator to submit unfinalized capsules,
- the admissibility of capsules originating from the revoked entity,
- the acceptance of capsules transported via the revoked transfer device.

Revocation does not require cooperation from offline entities and is enforced during capsule finalization and reception.

Previously finalized capsules remain verifiable but are subject to receiver policy.

5.15 Security Considerations and Security–Utility Trade-Off

This architecture prioritizes protection against:

- input-side exfiltration,

- unauthorized identity assertion,
- transit-phase manipulation,
- infrastructure-level compromise.

Key effects include:

- elimination of implicit execution paths,
- strict separation of identity authority and content access,
- revocation without offline trust leakage,
- containment under host or network compromise,
- absence of server-side profiling or behavioral metadata leakage.

At the same time, BEAP preserves cryptographic accountability: sender and receiver identities remain verifiably bound to each finalized capsule. In lawful or investigative contexts, this enables attribution and evidentiary analysis without reliance on hidden access mechanisms or centralized control.

5.16 Excursus: Extending Cryptocurrency Protocols via the BEAP Capsule Principle

The BEAP Capsule Principle and Protocol-Level Execution Authority

The BEAP capsule principle defines a security model that can be embedded directly into cryptocurrency protocols. Its core contribution is the cryptographic fixation of transaction intent and execution authority prior to execution, combined with non-authoritative transport and protocol-enforced execution bound to verifiable execution contexts. This model shifts trust away from discretionary software behavior and toward explicit, protocol-level validity rules.

In conventional hardware wallet-based cryptocurrency systems, cryptographic identity and private keys are device-bound, while transaction intent is typically assembled in external software and propagated through untrusted environments before signing. Although key custody may be secure, protocol validity is determined almost exclusively by possession of a private key. Once a valid signature is produced and broadcast, the protocol treats it as absolute authority, without regard to how intent was formed, under which conditions it was authorized, or whether the execution environment was compromised. As a result, intent integrity and execution correctness remain outside the protocol's trust boundary.

Applying the BEAP principle alters this boundary by introducing an explicit notion of execution authority as a protocol-level object. Transaction intent is first constructed in a canonical form and becomes authoritative only when bound into a capsule according to protocol rules. Capsule

unpackaging and execution on the receiver side are always treated as cryptographically authoritative operations and therefore require execution within hardware-attested environments under explicit sender-receiver handshake verification.

Sender-side attestation requirements are context-dependent. In privacy-preserving financial transactions, where capsule contents are encrypted and correctness of the ciphertext cannot be independently verified by validators, capsule sealing constitutes a cryptographically authoritative transformation. In such cases, capsule encryption and sealing must be performed within hardware-attested software to ensure intent integrity and prevent undetectable manipulation prior to execution. Where capsule construction remains canonical and publicly verifiable—such as non-encrypted or transparently committed capsules—sender-side hardware attestation is not strictly required for protocol validity and may be treated as a security hardening measure rather than a mandatory condition.

Off-Protocol PoAE for Legacy Blockchains

Intent-Bound Execution for Payments using BEAP Execution Capsules

Abstract

Blockchain platforms such as Ethereum, Bitcoin, and Solana provide reliable protocol-level execution semantics. Transactions are executed deterministically once they are correctly formed, signed, and accepted by consensus.

This document introduces Off-Protocol Proof-of-Authorized Execution (Off-Protocol PoAE), a complementary security model that binds human-verified intent to on-chain execution without modifying protocol semantics. The model follows a strict and practical sequence: explicit egress definition and isolation, followed by human authorization, on-chain execution enforcement, and verifiable execution completion.

Within this flow, BEAP Execution Capsules act as canonical authorization objects that can be validated directly by smart contracts and subsequently archived, correlated with execution results, and analyzed in software, enabling durable auditability of both authorized intent and completed execution.

pasted

1. Egress Definition and Isolation (First Principle)

1.1 Definition of Egress

In this model, *egress* refers to any outward interaction that could influence, redirect, or exfiltrate execution-relevant context, including:

- outbound network communication (RPC, HTTP(S), WebSockets),
- dynamic code loading or plugin execution,
- inter-process communication outside the trusted boundary,
- invocation of non-attested binaries or libraries.

Egress is treated as a concrete execution boundary, not an abstract policy.

1.2 Pre-Authorization Egress Isolation

Before a human reviews or authorizes intent:

- the authorization UI and logic run in a hardware-attested, source-available environment,
- outbound network access is restricted by default,
- only minimal, explicitly required local interfaces are permitted (e.g. hardware authentication input).

This isolation is intentionally scoped and time-limited. It can be enforced using standard application-level mechanisms (browser permissions, content security policies, or OS-level sandboxing) and does not require specialized infrastructure.

The objective is precise:

the human verifies intent in an environment that cannot silently redirect, substitute, or externally transmit authorization context.

No authorization is considered valid unless created under these conditions.

2. Authorization, Execution, and Completion as Distinct Concerns

Off-Protocol PoAE explicitly separates three concerns:

1. Authorization — what a human explicitly approved
2. Execution — the irreversible on-chain action
3. Completion — verifiable evidence that execution actually occurred

This separation is intentional.

Authorization captures *permission*.

Execution performs *state transition*.

Completion establishes *objective proof* that the authorized execution was realized.

3. Core Principle of Off-Protocol PoAE

Off-Protocol PoAE introduces authorization as an explicit, verifiable artifact that:

- exists prior to execution,
- is created under defined egress isolation, including:
 - mandatory hardware-backed second-factor authentication (2FA),
 - execution within a hardware-attested environment,
 - fully source-available and reproducible authorization code,
 - absence of cloud services, remote APIs, or hidden execution paths,
- and is validated at execution time.

Key properties:

- authorization, execution, and completion are temporally distinct,
- authorization is human-verified and hardware-backed,
- execution is permitted only if it matches the authorization exactly,
- completion is proven independently of encrypted intent content,
- enforcement is performed on-chain by a gatekeeping smart contract.

The authorization artifact itself:

- does not move value,
- does not execute logic,
- defines the permitted execution parameters.

4. Mandatory Two-Phase Human Verification

Human verification is a hard invariant.

4.1 Phase One — Authorization Test (Commit)

After egress isolation is established:

- recipient, asset, and bounds are presented in a hardware-attested, source-available UI,
- the user confirms intent using a hardware-backed authentication device,
- a formal authorization artifact is created.

This phase establishes what is allowed.

4.2 Phase Two — Execution Confirmation (Execute)

Immediately before execution:

- the user confirms again using the hardware-backed device,
- execution explicitly references the prior authorization,
- the smart contract verifies that execution parameters match the authorization exactly.

Both confirmations are mandatory.

5. Representing Authorization for Smart Contracts

Smart contracts cannot observe UI state, user explanations, or encrypted context. They operate exclusively on structured data and cryptographic proofs.

Off-Protocol PoAE therefore requires authorization to be represented as a machine-verifiable intent object.

This role is fulfilled by BEAP Execution Capsules.

6. BEAP in This Context: Execution Capsules

Within Off-Protocol PoAE, BEAP is used in a focused and constrained manner.

BEAP is not an orchestrator.

BEAP is not a cloud service.

BEAP is not a generic transport layer in this flow.

Here, BEAP defines:

a canonical execution capsule that encodes, signs, verifies, and preserves human-authorized intent, and enables verifiable correlation with completed on-chain execution.

7. Structure of a BEAP Execution Capsule

A BEAP Execution Capsule consists of two logically distinct sections.

7.1 Public / Enforceable Section

This section is:

- deterministic,
- readable by the smart contract,
- used directly for enforcement and execution binding.

Typical fields include:

- gatekeeper or smart-account address,
- recipient address,
- asset (native or token),
- exact amount or upper bound,
- permitted call type,
- chain identifier,
- nonce and expiry,
- canonical intent hash.

Required signatures:

- user signature,
- hardware-backed second-factor signature.

Only this section influences execution.

7.2 Encrypted Evidence Section (Optional)

This section is:

- encrypted (post-quantum secure),
- not processed by the smart contract,
- not used for execution decisions.

It may contain:

- human-readable intent descriptions,
- UI representation hashes,
- device attestation metadata,
- timestamps and authorization sequence,

- contextual references such as invoice identifiers.

Its purpose is evidence preservation, not enforcement.

7. Execution and Completion Flow Using BEAP Capsules

Prior to authorization, a cryptographic handshake establishes a trusted execution relationship between the authorization environment and the enforcing gatekeeper or protocol. This handshake binds hardware-backed authentication, compliant execution software, and protocol-defined verification rules into a stable trust context. Authorization artifacts generated outside this context are invalid by definition. All subsequent egress isolation, authorization finalization, and execution enforcement steps assume the presence of a valid handshake.

1. Egress isolation is established

2. Authorization phase

- Capsule is constructed and signed

3. Execution phase

- Smart contract is called with capsule, signatures, and execution calldata

4. On-chain verification

- Capsule integrity, signatures, nonce, expiry, and parameter matching are verified

5. Execution

- Value transfer occurs only if all checks succeed

6. Completion attestation

- The gatekeeping smart contract emits a deterministic execution record (e.g. event or receipt) that cryptographically binds the on-chain transaction to the capsule's canonical intent hash

Completion is therefore established by observable protocol artifacts, not by encrypted capsule contents.

9. Archival and Post-Execution Analysis

After execution, BEAP execution capsules and their correlated on-chain execution records can be persistently archived.

Archived artifacts may be:

- indexed and searched,

- correlated with transaction receipts and logs,
- analyzed for behavioral or compliance patterns,
- re-verified independently at any later point in time.

Because the capsule format and intent hash derivation are canonical and source-available, archival analysis does not depend on proprietary tooling or trusted intermediaries.

10. Explicit Limits

Off-Protocol PoAE does not:

- secure transactions executed outside the gatekeeper boundary,
- prevent deliberate user bypass,
- alter blockchain protocol behavior,
- claim that encrypted intent implies execution.

It does:

- bind execution to human-verified intent,

- prevent parameter substitution,
 - provide deterministic, auditable authorization and completion semantics.
-

11. Conclusion

Protocol-level execution semantics remain unchanged.

Off-Protocol PoAE introduces a structured model that begins with explicit egress definition and isolation, binds authorization to human verification, enforces intent at execution time, and establishes verifiable proof of completion using native blockchain artifacts.

In this flow, BEAP Execution Capsules provide the formal, verifiable representation of intent, while on-chain execution records provide objective completion evidence—together enabling enforcement, archival, and analysis without modifying existing blockchain protocols.

This is not orchestration.

This is not cloud security.

It is a precise authorization and completion primitive for intent-bound execution.

Handshake Establishment via BEAP Capsules under WRGuard Constraints

Proof of Authorized Execution (PoAE)

1. Problem Statement: What Existing Systems Cannot Prove

Blockchain systems provide strong guarantees about ledger integrity and deterministic execution, but they remain largely silent on a critical question:

Was this irreversible action deliberately authorized by a human, under bounded and verifiable conditions?

In current systems:

- possession of a private key is sufficient to trigger irreversible execution,
- authorization and execution are conflated,
- human intent is inferred, not proven,
- execution environments are opaque to validators,
- custodial platforms enforce security through discretionary, off-protocol processes.

Wallet UX, multisignature schemes, MPC, withdrawal delays, and application-level 2FA are compensating controls. They reduce risk but do not create protocol-verifiable execution authority.

PoAE addresses this missing layer.

2. Core Definition of Proof of Authorized Execution

Proof of Authorized Execution (PoAE) is a cryptographic validity proof attached to a state transition that demonstrates that an action was:

1. explicitly finalized by a human,
2. finalized within a non-expandable authorization scope, and
3. executed strictly within that scope.

PoAE separates cryptographic key possession from execution authority.

Unlike a transaction signature—which merely attests that a key was used—PoAE attests that execution authority itself was deliberately finalized under bounded conditions prior to execution.

The presence of a valid PoAE becomes a necessary condition for execution validity.

3. Execution Authority vs. Transactions

A central property of PoAE is that finalization applies to execution authority, not to individual transactions.

An authorization event:

- fixes the *maximum* permitted execution scope,
- may authorize one or multiple executions,
- cannot be expanded, modified, or reused outside its bounds.

Authorized scope may include:

- recipient or target identity,
- maximum transferable value or action bounds,
- validity window,
- permitted action class.

Once finalized, this scope is non-expandable. Any execution exceeding it is deterministically rejected.

4. Hardware-Bound Human Finalization

Execution authority finalization must be performed using a hardware-bound secondary authentication factor that satisfies a protocol-defined proof class.

Validators do not depend on vendors or devices. They verify that the authorization proof:

- is derived from a non-exportable hardware-bound secret,
- is fresh and non-replayable,
- is cryptographically bound to a canonical intent digest,
- is independent of the primary execution environment,
- is verifiable under deterministic rules.

Authorization proofs failing this class are invalid regardless of transaction signature validity.

This elevates 2FA from a UX feature to a cryptographic execution-authority primitive.

5. Handshake as the Root of Trust

Before authorization can occur, a cryptographic handshake establishes a trusted execution context.

Handshakes are established using BEAP capsules, processed under WRGuard principles. These capsules:

- are admitted in sanitized, non-executable form,

- are cryptographically verified before trust is granted,
- bind authorization software, hardware factors, and verification rules.

Authorization artifacts produced outside a valid handshake context are invalid by definition.

The handshake does not authorize execution.

It establishes who is capable of authorizing.

6. Egress Control as a Hard Boundary

Authorization must occur in an environment where egress is explicitly bounded.

Egress includes:

- outbound network communication,
- dynamic code loading,
- execution of non-attested binaries,
- any channel capable of altering execution semantics post-authorization.

PoAE does not attempt to observe runtime behavior. Instead, it enforces proof classes that can only be produced by environments satisfying these constraints.

This prevents silent redirection, substitution, or exfiltration of authorization context.

7. Dual Human Confirmation (Commit / Execute)

PoAE requires two distinct human confirmations, both after handshake and egress isolation:

7.1 Authorization Commit

The human finalizes execution authority, fixing scope and intent.

7.2 Execution Confirmation

Immediately before execution, the human confirms the concrete execution attempt, binding it to the previously finalized authorization.

Authorization without execution confirmation does not move value.

Execution without prior authorization is invalid.

This closes timing, replay, and substitution gaps.

8. Validator Enforcement Model

PoAE enforcement is deterministic and non-programmable:

- no smart contracts,
- no subjective logic,
- no upgradeable enforcement paths.

Validators verify:

- existence and validity of PoAE,
- binding to canonical intent,
- hardware-bound proof compliance,
- strict scope adherence.

If any check fails, execution is invalid.

9. Custodians and Intermediaries

PoAE applies uniformly to end users and intermediaries.

Custodial platforms may hold keys and operate internal systems, but on-chain execution authority is no longer discretionary.

Any on-chain transfer—withdrawal, deposit, rebalancing—must satisfy PoAE validity conditions.

Possession of private keys alone is insufficient.

10. BEAP Execution Capsules and Completion Evidence

PoAE uses BEAP Execution Capsules as canonical execution artifacts.

In addition to enforceable public fields, execution capsules MAY contain an encrypted evidence section including:

- successful execution details,
- transaction identifiers and receipts,
- timestamps and confirmation metadata.

This encrypted section:

- does not influence execution,

- serves auditability, archival, and analysis,
 - enables post-execution inspection without weakening enforcement.
-

11. Loss and Recovery of Execution Authority

Loss of a secondary authentication factor does not require central recovery mechanisms.

Recovery is achieved through:

- reinstallation of compliant authorization software,
- re-enrollment of a hardware-bound factor,
- reestablishment of handshake,
- creation of a new authorization finalization event.

Previously finalized scopes cannot be reused or expanded.

Recovery restores *the ability to authorize*, not past authority.

12. Applicability Beyond Cryptocurrency

PoAE is not limited to blockchain or cryptocurrency systems.

It applies to any domain where:

- actions are irreversible or security-critical, and
- execution must be conclusively attributable to deliberate human authorization.

Examples include:

- financial settlement and approval workflows,
- industrial automation,
- IoT device control and firmware updates,
- secure document release,
- controlled AI and automation systems.

Across domains, the invariant holds:

Execution authority must be finalized by deliberate human action under non-expandable constraints, and execution must be verifiably bound to that authority.

13. Conclusion

PoAE introduces a missing primitive: verifiable execution authority.

Authorization is no longer inferred from key possession.

Execution is no longer assumed to be deliberate.

Trust is no longer delegated to applications or custodians.

Instead, PoAE provides:

- explicit human finalization,
- bounded authority,
- protocol-verifiable enforcement,
- auditable completion.

This is not multisig.

This is not MPC.

This is not application-level 2FA.

It is a general execution-authority model for irreversible actions.

Auditable, Post-Quantum-Ready PoAE Trace

For auditability, the handshake is treated as a first-class, verifiable artifact within the PoAE lifecycle. Handshakes are established using BEAP capsules processed under WRGuard principles, producing a deterministic, integrity-protected reference that anchors a trusted authorization context. Subsequent authorization and execution capsules explicitly bind to this reference, allowing the final BEAP logging capsule to contain a complete, causally verifiable trace from trusted environment establishment through authorization to successful execution.

The PoAE process is post-quantum ready by design. All integrity- and authority-critical bindings across the handshake, authorization, execution, and receipt path are defined in a way that allows the use of post-quantum-secure signature schemes. Crucially, a blockchain can only be considered meaningfully post-quantum secure if the entire transfer, authorization, and execution chain preserves equivalent guarantees. If value transfer or execution can be influenced through non-post-quantum-secure components, ledger-level post-quantum security alone is insufficient.

BEAP capsules are designed specifically to address this requirement by providing a transport-agnostic, verifiable container for post-quantum-ready authorization and execution artifacts, ensuring that integrity and provenance remain independent of underlying transport layers. Final BEAP logging capsules may be post-quantum encrypted by policy to ensure long-term confidentiality of execution metadata while preserving deterministic, verifiable linkage across all stages of the PoAE trace.

To fully close the post-quantum security gap, hardware-backed components must evolve accordingly. Hardware keys, secure elements, and hardware attestation mechanisms used for authorization and execution confirmation must themselves be post-quantum ready to prevent downgrade paths. PoAE therefore deliberately specifies cryptographic proof classes rather than concrete devices or algorithms, allowing hardware-backed authentication and attestation mechanisms to be replaced or upgraded with post-quantum-secure implementations as they become available.

6. System-Generated Visual Envelope

To avoid empty-message semantics and provide user clarity, BEAP defines a visual envelope generated entirely by the client.

The visual envelope (eg wr code):

- is not transmitted,
- is not derived from container bytes,
- is constructed solely from verified metadata and admission state.

It conveys:

- presence of a BEAP message (can be generic to avoid spam filters),

- verification status,
- sender identity,
- container identifier,
- permitted user actions.
- Permitted receiver / role / policy

The envelope is a UI object, not a file.

7. User Interface States

7.1 Verified Message

After successful admission, the client MAY present:

- reconstructed or parsed message text derived from the verified Capsule,
- safe attachment representations (e.g. parsed text or rasterized previews),
- provenance indicators (verified sender, container hash).

Displayed content is always derived, never rendered from transport artifacts.

7.2 Verification Failure

If identity or integrity verification fails:

- no content is rendered,
- a clear failure notice is displayed,
- optional diagnostic metadata may be shown.

7.3 Unsupported or Malformed Container

If the container cannot be processed:

- no content is rendered,
- a compatibility notice is displayed.

8. Explicit Export, Forwarding, and Redirection

Non-rendered processing does not prohibit transfer.

After admission, clients MUST allow explicit, user-initiated or policy-controlled actions to:

- export the BEAP container as a single file,
- forward the container unchanged,
- archive or store the container,

- re-wrap content into a new BEAP container under a new sender identity.

All such operations transfer the same verified byte sequence without interpretation. Any modification is immediately detectable.

9. Spam Compatibility and Infrastructure Integration

BEAP fully conforms to established email and messaging hygiene:

- a minimal, human-readable wrapper message is included,
- SPF, DKIM, and DMARC alignment are required for email transports,
- message and attachment metadata are deterministic and standards-compliant.

Spam filters and classifiers evaluate the outer transport message, not the BEAP container. BEAP therefore integrates cleanly with existing mail servers, gateways, and messaging systems.

10. Application to Messaging Systems

BEAP semantics apply equally to:

- email,
- enterprise chat systems,

- instant messengers,
- collaboration platforms.

Messaging transports carry the BEAP container as a structured message or attachment. Clients perform the same identity-first admission process prior to rendering any content.

Messaging UIs may display a visual BEAP envelope within the chat timeline, followed by verified, reconstructed content only after admission.

11. Security Properties

The BEAP envelope model provides a strong security framework:

- Spoofing resistance through identity-first admission.
- Tamper and replay detection via container-wide cryptographic binding.
- Elimination of active-content attacks by prohibiting implicit rendering.
- Deterministic verification through schema-bound checks.
- Clear trust boundaries between transport and presentation.

Trust is established before presentation, not inferred afterward.

12. The exclusion of compression is necessary to enable exhaustive, deterministic schema validation at admission time; introducing decompression would create a non-transparent transformation step capable of concealing structure and violating the protocol's closed verification model. Compressed archives are incompatible with BEAP. Decompression is a security risk and violates BEAP's deterministic, non-rendered verification model.

13. Conclusion

BEAP treats email and messaging systems as secure transports for verifiable containers rather than as document-rendering channels. By enforcing a single opaque container, identity-first admission, non-rendered processing, and client-generated visual envelopes, BEAP achieves interoperability with existing infrastructure while delivering high-assurance security. Every visible element is either cryptographically verified or explicitly marked as untrusted by construction, enabling secure communication, redistribution, and automation across heterogeneous transports.

Cryptographic Model of BEAP Capsules and the WRCode System
(Post-Quantum, Identity-First, Encrypted by Design, Tiered Trust)

This section defines the cryptographic foundations of BEAP (Bidirectional Email Automation Protocol) Capsules and the extended mechanisms provided by the WRCode system.

The architecture is post-quantum from inception and mandates cryptographic protection as a core protocol requirement. Confidential payloads are always protected using mandatory hybrid encryption (classical and post-quantum in parallel), with no downgrade path or fail-open fallback. Public WR Codes are intentionally unencrypted and tamper-proof to preserve transparency and auditability. The use of mandatory parallel hybrid encryption does not introduce protocol-level latency, as cryptographic operations are non-interactive and amortized outside of critical execution paths. The protocol allows specific encryption methods to be exchanged and updated without altering the security model, ensuring the cryptographic profile can remain state of the art over time.

BEAP is designed for secure, bidirectional automation over email in hostile content environments. It enforces identity-first verification, deterministic cryptographic validation, strict fail-closed behavior, and cryptographically bounded interpretation. WRCode extends BEAP with optional mechanisms for public auditability, tamper-proof automation, and long-term provenance, particularly for automation executed on user devices via visually and cryptographically classified WR Codes.

The BEAP identity signature functions as a Merkle-root-equivalent over payload, structure, identity, context, and policy signals, ensuring that any semantic or structural modification is cryptographically detectable; higher-level indicators such as origin classification, relationship or handshake state, and

sender or user type are not independent trust inputs but deterministic derivations from the signed signal set.

As of today, most widely deployed hardware attestation mechanisms (e.g., TPMs and HSMs) rely on classical cryptographic algorithms that would not be secure against powerful quantum adversaries; while post-quantum primitives are emerging in new hardware designs, post-quantum-secure attestation is not yet ubiquitous in deployed hardware systems.

BEAP capsule generation solely relies on hardware attestation as its root of trust. As of today, widely deployed hardware attestation mechanisms are still based on classical cryptographic primitives and are not post-quantum secure. Consequently, even if a BEAP capsule is structured to support post-quantum cryptographic algorithms, it cannot be considered post-quantum secure at generation time, because the underlying hardware attestation does not yet provide post-quantum guarantees. In this sense, BEAP is architecturally prepared for post-quantum security, but cannot be claimed to be post-quantum secure in practice until hardware attestation itself becomes post-quantum ready.

1. Scope and Architectural Separation

Cryptographic responsibilities are strictly separated:

BEAP defines:

- Capsule structure

- Mandatory post-quantum encryption
- Deterministic hashing
- Post-quantum digital signatures
- Identity-first cryptographic admission & handshake detection
- Fail-closed validation
- Secure, bidirectional automation over email
- No dependency on blockchains or external registries

WRCode extends BEAP with:

- Cryptographic classification of WR Codes by execution scope
- Public template publication for published WR Codes
- Optional global anchoring
- Externally verifiable timestamps
- Anchored template registries
- Optional anchored execution log roots

BEAP Capsules are valid, verifiable, and complete without WRCode.

WRCode never replaces or weakens BEAP's cryptographic structure and only adds execution-scope-specific identity, auditability, and provenance where required.

2. Identity-First Admission Gateway

Every BEAP Capsule MUST pass sender identity verification and handshake verification as the first mandatory processing step.

Before any Capsule is decrypted, parsed, or semantically interpreted, the receiver MUST cryptographically verify the sender identity and hardware-attestation status using a minimal, fixed-format admission envelope.

The admission gateway enforces:

- cryptographic attribution of the Capsule to a sender identity appropriate for its execution scope
- protocol and version identification
- strict size limits
- cryptographic authentication of the enclosed Capsule ciphertext or its cryptographic hash

Capsules originating from unknown, unauthenticated, suspended, revoked, or scope-incompatible identities are rejected immediately and never reach depackaging, decryption, or template resolution.

This identity-first gateway removes anonymous input, prevents spoofing, and ensures all Capsules are attributable before any complex processing occurs.

3. Encryption and Confidentiality

3.1 Protocol Requirement

BEAP Capsule encryption is a policy-gated capability restricted to hardware-attested senders and receivers. Capsules generated by non-attested senders MUST remain unencrypted, and encrypted Capsules MUST only be admitted by hardware-attested receivers.

Two explicit exceptions apply:

- (1) WR Codes used in closed enterprise environments, where all producers and consumers operate under the same administrative and attestation domain.
- (2) WR Codes for private use, where Capsules are consumed exclusively by the originating device or user and no external party can scan or access the content.

In both cases, encrypted Capsules are permitted without cross-party attestation because the content is consumed only by entities that also produced it, and no third-party admission occurs.

3.2 Post-Quantum Key Establishment

Confidentiality is established using a post-quantum Key Encapsulation Mechanism (KEM).

Typical flow:

1. The sender encapsulates a shared secret using the recipient's public key.
2. A symmetric Content Encryption Key (CEK) is derived via a deterministic KDF.
3. The Capsule payload is encrypted using authenticated symmetric encryption.

Only post-quantum KEMs (e.g. ML-KEM / CRYSTALS-Kyber) are permitted.

3.3 Symmetric Encryption

The encrypted Capsule payload uses authenticated encryption with associated data (AEAD), such as AES-256-GCM or ChaCha20-Poly1305.

Symmetric encryption protects:

- Capsule fields
- Template references
- Attachment descriptors
- Policies and metadata

Ciphertext integrity is enforced by AEAD authentication and Capsule signatures.

4. WR Code Classification and Execution Scope

WR Codes are cryptographically classified by execution scope.

Identity requirements, auditability, and execution privileges are proportional to the scope in which a WR Code may execute.

4.1 Private Personal WR Codes can be encrypted

- Created by individual users for non-commercial use
- Bound to local or paired device identities
- No DNS verification required
- Not published in public registries
- Executable only on the creator's devices or explicitly paired environments
- Visually distinct from other WR Code classes

Private personal WR Codes never execute in public or unpaired environments.

4.2 Internal WR Codes are encrypted

- Used within organizations or closed groups
- Bound to internal organizational identities or PKI
- No public DNS verification required
- Not publicly discoverable
- Executable only within authorized internal environments
- Visually distinct from personal and public WR Codes

Private internal WR Codes cannot be executed outside their defined organizational scope.

4.3 Public WR Codes

Public WR Code Payload Transparency

For WR Codes classified as public, the Capsule payload:

- Is intentionally transmitted unencrypted
- MUST remain cryptographically authenticated
- MUST remain tamper-proof
- MUST remain identity-bound

Public WR Code Capsules are designed to be:

- fully inspectable
- publicly auditable
- deterministic
- safe by construction

They do not carry secrets, credentials, or private data.

- Publicly distributable and scannable by unpaired user devices
- Bound to DNS-verified, registry-listed sender identities
- Associated with paid WR Accounts
- Subject to public template publication and auditability requirements
- Visually distinct from private WR Code classes

Public WR Codes represent the highest trust exposure and therefore require the strongest identity attribution and auditability.

WR Code classification is cryptographically bound and cannot be altered without invalidating verification.

5. Template Hashing and Semantic Binding

Before constructing a Capsule, the sender computes a deterministic hash of the template used to structure the Capsule:

```
template_hash = SHA-256(template)
```

The template hash and template identifier are embedded into the encrypted Capsule payload.

Upon receipt, the receiver recomputes the hash over the verified template.

If the computed hash does not match the embedded reference, the Capsule is rejected.

This prevents template substitution, silent semantic changes, and structural downgrade attacks.

6. Capsule Signature (Post-Quantum)

After identity verification and decryption, the sender's identity is cryptographically bound to the Capsule contents via a post-quantum digital signature:

```
capsule_digest = SHA-256(capsule_payload_without_signature)
```

```
signature = ML-DSA.Sign(sender_private_key, capsule_digest)
```

Verification:

```
ML-DSA.Verify(sender_public_key, signature, capsule_digest)
```

Only post-quantum signature algorithms are permitted.

The signature covers all structured fields, metadata, hashes, template references, attachment references, declared policies, and execution-scope indicators.

7. Fail-Closed Validation Model

Admission of a BEAP Capsule or WR Code–derived artifact is permitted only if all applicable checks succeed:

- Sender identity verification appropriate to the WR Code class and deployment context succeeds
- Cryptographic decryption succeeds *if and only if encryption is present and permitted by policy*
- Post-quantum and classical signature verification succeeds for all declared signatures
- Referenced template hashes match locally verified and policy-approved templates
- Execution scope and capability constraints (permissions, environment, side-effects) are satisfied for the receiving context

Any failure results in immediate rejection without fallback, partial acceptance, or degraded behavior.

8. Attachment Binding and Cryptographic Boundaries

BEAP Capsules do not embed large binary attachments directly.

Attachments are cryptographically bound via encrypted and signed descriptors containing:

- unique attachment identifiers
- cryptographic hashes
- declared size and type
- access and handling policies
- optional rasterized preview references

Attachment binaries may be transmitted separately but remain within the BEAP trust boundary through hash binding and signature coverage. Decryption of attachment binaries is explicitly disallowed within WRGuard-protected environments and, when permitted by policy, can only occur outside the WRGuard-protected network in an isolated context. Decrypted attachment binaries are never directly accessible or processed inside the protected environment.

10. Public Execution Without Prior Handshake (Corrected)

Public execution is supported only for WR Codes classified as public. For public WR Codes, Capsules/Envelope contents are not encrypted to preserve transparency and auditability. Public execution occurs in a restricted, sandboxed environment on the user device and is treated as untrusted by default.

In such cases:

- sender identity remains cryptographically attributable (signature-bound)
- authenticity and integrity verification are mandatory (schema, hashes, signatures, template hashes)
- execution does not imply trust or authorization and remains non-privileged and capability-bounded
- explicit user consent is always required before any execution or side-effect
- any access to local secrets, accounts, or privileged resources requires a prior handshake / explicit authorization path (not the public path)

Private WR Codes MUST NOT be executed via public execution paths. They require a private admission context (handshake and/or controlled domain policy) and are not discoverable/executable through public scanning.

11. Extended Cryptographic Model in the WRCode System

WRCode optionally provides:

- template anchoring on Solana
- template anchoring on IOTA
- anchoring of execution log roots

These mechanisms add external verifiability and long-term provenance but are not required for BEAP correctness.

12. Summary

BEAP provides:

- post-quantum encryption
- identity-first cryptographic admission
- post-quantum Capsule signatures
- deterministic template hashing
- strict fail-closed validation

- cryptographic binding of external attachments
- secure, bidirectional email automation

WRCode extends BEAP with:

- cryptographically enforced WR Code classification
- proportional identity requirements by execution scope
- optional public template publication
- optional global anchoring
- externally verifiable provenance

BEAP ensures post-quantum confidentiality, authenticity, and integrity.

WRCode enables controlled, auditable, and scope-appropriate automation across private and public environments.

BEAP may support an optional “sealed originals” mode in which original email context and attachments are transmitted as separately encrypted, content-addressed blobs. The Capsule binds each blob via signed descriptors and cryptographic hashes, preserving identity-first admission and deterministic verification while enabling end-to-end confidentiality for originals without embedding large binaries into the Capsule.

Hardening of BEAP Capsule Depackaging

Depackaging is the most security-critical phase in any message-based protocol, as it involves processing attacker-controlled input. BEAP explicitly treats depackaging as a high-risk boundary and applies layered hardening measures to ensure that malformed or adversarial Capsules cannot lead to implicit execution, privilege escalation, or silent failure.

The goal of BEAP depackaging is not permissive handling, but deterministic rejection or strictly bounded acceptance under explicit conditions.

Identity-Gated Entry

BEAP enforces identity-first admission.

No Capsule reaches the depackaging stage unless the sender identity has already been cryptographically verified and attributed to a persistent, non-anonymous identity appropriate to the WR Code's execution scope. This removes anonymous attack traffic entirely and prevents large-scale probing, fuzzing, or spray-style attacks against the depackaging logic. Receiver identity and role are used to determine applicable depackaging and decryption rules.

Minimal Trusted Computing Base

Depackaging is strictly separated from interpretation and execution logic.

The depackaging component is designed to:

- operate on fixed-size or strictly bounded inputs
- perform no semantic interpretation beyond verification
- execute no side effects (no file writes, no network calls, no automation)
- fail closed on any inconsistency

Only cryptographic verification, deterministic parsing, and structural validation are permitted at this stage.

Strict Bounds and Deterministic Parsing

All depackaging operations enforce hard limits, including:

- maximum Capsule size
- maximum field lengths
- fixed protocol versions
- bounded attachment descriptors

There is no best-effort parsing, no recovery from malformed input, and no partial acceptance. Any deviation results in immediate rejection.

Cryptographic Ordering

Depackaging follows a strict verification order:

1. Sender identity verification
2. Receiver identity verification (if applicable)
3. Ciphertext integrity verification
4. Post-quantum decryption (if encryption is applied)
5. Capsule signature verification
6. Template hash verification

No later step is executed unless all prior steps succeed. This prevents downgrade attacks, semantic confusion, and misuse of partially verified data.

No Implicit Execution Surface

BEAP Capsules contain no executable code.

Depackaging never triggers rendering, scripting, macro evaluation, or attachment execution.

Attachments are cryptographically bound but handled externally and only after explicit user consent and policy enforcement.

Containment and Operational Hardening

Implementations are expected to execute depackaging logic in constrained environments using:

- memory-safe implementations
- strict resource limits
- process isolation or sandboxing
- extensive fuzz testing of the depackaging boundary

These measures ensure that residual implementation risks are contained, observable, and non-silent.

Worst-Case Consequences of a Successful Depackaging Attack

Even under extreme assumptions—such as a previously unknown vulnerability in the depackaging logic—the maximum plausible impact is deliberately limited by BEAP’s design.

In the worst-case scenario, a successful attack could result in:

- a denial of service affecting the depackaging component
- a crash or forced restart of the verification process
- rejection or loss of availability for incoming Capsules

Crucially, even in this scenario:

- no implicit code execution is triggered
- no automation logic is executed
- no credentials or secrets are released by design
- no attachment content is executed or rendered
- no privilege escalation occurs within the system

All actions remain attributable to a verified sender identity, allowing immediate revocation, blocking, and forensic analysis.

BEAP does not attempt to eliminate all theoretical implementation risk. Instead, it ensures that any successful exploitation degrades into contained failure rather than silent compromise.

Security Outcome

Through identity-first admission, strict cryptographic ordering, deterministic parsing, and fail-closed behavior, BEAP transforms depackaging from an implicit execution risk into a bounded verification step with limited blast radius.

Attackers are forced to reveal themselves early, operate under attribution, and accept that even a successful exploit attempt yields minimal impact while incurring high operational and reputational cost. This matches the expectations of hardened enterprise communication systems and shifts the threat model from silent compromise to accountable, observable failure.

Identity-Derived Admission Signals and Relationship-Aware Handling

BEAP and the WRCode system MAY evaluate additional identity-derived admission signals as part of the identity-first admission process. These signals are derived exclusively from the verified sender identity and associated metadata and are not independent trust primitives.

Such signals provide context for handling decisions, not authorization, and do not replace cryptographic identity verification or Capsule validation.

Origin Context Signal

An origin context signal MAY be included as part of the identity-bound admission metadata. The signal represents a coarse-grained origin classification, such as a geographic or jurisdictional region (e.g. EU, North America, South Asia), derived from verifiable account, registry, or infrastructure attributes.

The origin context:

- is coarse-grained and non-identifying
 - does not rely on real-time network location
 - is cryptographically bound to the sender identity
 - does not imply trust or legitimacy
-

Handshake and DNS Proof as Supplemental Signals

In enterprise and organizational deployments, handshake state and DNS-verified organizational control MAY be evaluated as supplemental admission signals.

These signals indicate:

- whether a prior bilateral relationship exists
- whether the sender demonstrates control over a stable organizational domain
- whether the request aligns with expected operational relationships

DNS proof and handshake state do not grant trust on their own and do not override cryptographic verification. They provide additional context for relationship-aware handling.

Signal Usage and Handling Decisions

When combined with identity-first admission, these identity-derived signals MAY be used to support:

- policy-controlled admission filtering
- contextual warnings during user consent
- detection of unexpected origin or relationship changes
- additional friction for cross-region or out-of-relationship automation attempts without established handshakes

These signals help determine whether an automation request:

- aligns with known and expected relationships
- should be handled inside the operational network
- should be inspected only in isolated or offline environments
- or should be ignored entirely

The final decision always remains with policy configuration and explicit user consent.

Security Outcome

By evaluating origin context, handshake state, and DNS proof as identity-derived signals, BEAP and WRCode reduce realistic attack vectors without introducing implicit trust or automatic execution.

The system ensures that automation requests lacking a clear relationship context are never forced into production environments, reinforcing the identity-first security model while preserving flexibility, user autonomy, and protocol simplicity.

Deterministic Byte-Level Chunking as a Security Primitive in Secure Automation Envelopes

Abstract

Secure automation envelopes transporting structured semantic payloads and encrypted original artifacts must operate under adversarial conditions while traversing heterogeneous and partially untrusted infrastructures. This article presents deterministic byte-level chunking as a core architectural security mechanism for large JSON payloads and encrypted artifacts in BEAP-style envelope systems. Beyond transport resilience, chunking enables early anomaly detection, non-materialized processing, bounded sandbox execution, and fail-closed behavior. When combined with identity-bound hash aggregation and hardware-attested senders, chunking significantly reduces the

temporal and spatial attack surface available to malicious payloads. A fundamental constraint is emphasized throughout: chunking applies strictly to bits and bytes, never to characters or encoding abstractions, preserving canonicality and cryptographic determinism.

1. Introduction

Automation envelopes increasingly transport:

1. Structured semantic payloads, typically encoded as JSON capsules.
2. Original binary artifacts, such as PDFs, office documents, images, or logs, commonly encrypted and subject to strict policy controls.

While these payloads form a single logical unit, real-world transport layers introduce size limits, buffering behavior, scanning interference, and partial delivery failures. At the same time, secure automation systems must defend against malformed inputs, resource exhaustion, delayed tampering detection, and post-decryption exploitation.

Deterministic chunking addresses these challenges not merely as a transport strategy, but as a security primitive embedded into the protocol itself.

2. Byte-Level Chunking as a Structural Requirement

2.1 Chunking at the Bit and Byte Level Only

A strict design rule is that chunking operates exclusively on raw bits and bytes.

This constraint is mandatory because:

- JSON payloads are typically UTF-8 encoded, where characters may span multiple bytes.
- Character-level chunking introduces ambiguity and encoding-dependent behavior.
- Cryptographic hashing, signature verification, and authenticated encryption operate on byte sequences, not textual abstractions.

Accordingly:

- Chunk boundaries are defined by byte offsets.
- Any canonicalization occurs *before* chunking.
- Reassembly is a deterministic byte concatenation process.

This ensures that integrity verification remains stable across implementations and execution environments.

3. Non-Materialized Processing and Early Rejection

3.1 Chunks Are Not Materialized

A key security property of the design is that chunks are not fully materialized into memory during initial validation.

Instead:

- Chunks are processed incrementally.
- Hashes are computed and compared as data streams through bounded buffers.
- No complete payload exists in memory prior to validation success.

This sharply limits the attacker's ability to exploit parser behavior, memory layout, or delayed validation paths.

3.2 Out-of-Band Metadata and Early Hash Validation

Chunk metadata—such as chunk count, sizes, and expected hashes or Merkle roots—is stored and verified outside the chunk data itself, typically in a signed manifest.

The validation sequence is:

1. Verify envelope identity and manifest authenticity.
2. Validate declared chunk structure and limits.
3. Hash incoming chunks incrementally.
4. Compare hashes immediately against declared values.

Any mismatch results in immediate abort and fail-closed termination.

This approach enables:

- Early detection of tampering or truncation.
- Rapid shutdown before deeper parsing or decryption occurs.
- Minimal exposure even under active attack.

4. Chunk Hashing and Identity-Bound Aggregation

4.1 Per-Chunk Hashing

Each chunk is associated with an independent cryptographic hash:

$\text{chunk_hash}_i = H($

```
chunk_index ||  
chunk_length ||  
raw_bytes  
)
```

This construction:

- Binds chunk position to content.
- Detects reordering, duplication, or omission.
- Enables incremental verification without full payload assembly.

4.2 Merkle-Root-Equivalent Binding

Chunk hashes are aggregated using a Merkle tree or equivalent ordered hash structure, producing a single root hash.

This root hash is bound into the envelope's identity signature together with:

- Sender identity
- Policy identifiers

- Origin and handshake metadata
- Payload type and versioning information

Any deviation at the chunk level propagates deterministically to identity verification failure.

5. Security Implications for Encrypted Payloads

5.1 Hardware-Attested Senders and Defense in Depth

In the described architecture, encrypted capsules originate from hardware-attested software environments. This significantly reduces the likelihood of malicious payload generation at the source.

However, receiver-side chunk hashing remains valuable because:

- Attestation does not eliminate transmission-time manipulation.
- Compromise scenarios must be assumed at all trust boundaries.
- Defense-in-depth requires independent verification stages.

Chunk hashing therefore acts as an additional, orthogonal security layer.

5.2 Reduced Exploitation Window Post-Decryption

Encrypted chunks are decrypted only after:

- Identity verification
- Chunk hash validation
- Policy admission

Because decryption occurs chunk-by-chunk within bounded buffers, the time window during which decrypted data exists in memory is extremely narrow.

This significantly limits:

- Payload-driven exploitation attempts.
- Side-channel observation opportunities.
- Post-decryption parser abuse.

6. Bounded Sandbox Execution and Fail-Closed Enforcement

All parsing, hashing, and optional decryption occurs within sandboxed execution environments with:

- Strict memory limits

- CPU and time quotas
- No implicit recovery paths

If any abnormality is detected—hash mismatch, size violation, timeout, or unexpected structure—the process terminates immediately and fails closed.

This behavior applies equally to:

- Structured JSON payloads
- Decrypted original artifacts
- Derived representations such as rasterized or parsed text

7. Separation of Safe Representations and Originals

Chunking reinforces a strict architectural separation:

- Capsule payloads contain structured, deterministic representations used by automation logic.
- Original artifacts remain encrypted, chunked, and policy-gated.

Automation systems never operate directly on original binaries, reducing exposure even when originals must be preserved for archival or compliance reasons.

8. Risks of Non-Chunked and Fully Materialized Designs

Systems that process monolithic payloads without chunking are exposed to:

- Delayed integrity detection.
- Unbounded memory allocation.
- Extended post-decryption exposure.
- Higher susceptibility to parser and sandbox escape techniques.

Deterministic chunking mitigates these risks by enforcing early validation, bounded execution, and explicit assembly rules.

9. Design Constraints and Best Practices

A secure envelope specification should mandate:

- Byte-level chunking only.
- Non-materialized incremental processing.
- Manifest-first validation.
- Explicit limits on chunk size, count, memory, and execution time.

- Immediate fail-closed termination on any anomaly.
- No compression and no opaque container formats.

These constraints support auditability, interoperability, and predictable security behavior.

In addition to cryptographic hash verification, optional bit-level consistency checks MAY be provided as out-of-band metadata (e.g. declared bit length or padding properties). These signals are evaluated prior to or alongside hash validation to enable early anomaly detection and fail-closed termination, and MUST NOT modify or reinterpret the underlying byte sequence.

10. Conclusion

Deterministic byte-level chunking functions as a core security primitive in secure automation envelopes. By preventing full materialization, enabling early anomaly detection, constraining sandbox execution, and minimizing post-decryption exposure, chunking substantially reduces the attacker's available surface and time window. When combined with hardware-attested senders and identity-bound hash aggregation, this approach forms a layered, resilient foundation for transporting large structured payloads and encrypted originals across untrusted infrastructures.

Threat Model for BEAP Capsules and the WR Code System

(Post-Quantum Signatures, Public Auditability, Policy-Gated Encryption, Consent-Bound Execution)

Normative Clarification (Applies Globally)

Publicly executable WR Codes may be scanned without a prior handshake. However, no automation step is ever performed without explicit, user-mediated consent on the receiving device. Verification, parsing, inspection, and pre-execution analysis do not imply trust, authorization, or execution.

Public WR Codes are never encrypted.

Their Envelopes and Capsules remain fully unencrypted to preserve transparency, auditability, and public verifiability.

Capsule encryption is a policy-gated capability that applies only to non-public WR Codes and is permitted exclusively:

- between hardware-attested senders and hardware-attested receivers, or
- within explicitly defined closed enterprise environments, or
- in private-use contexts where Capsules are consumed only by the entities that produced them and no third party can scan or access the content.

Encrypted Capsules are never admissible through public execution paths.

Risk 1 — Capsule Tampering in Transit

Risk:

An attacker modifies Capsule fields, policies, template references, attachment descriptors, or ciphertext during transport.

Mitigation / Solution:

- Deterministic canonicalization of all security-relevant fields prior to hashing and signing
- Mandatory Capsule signature verification covering Capsule structure, policies, template references, and attachment descriptors
- Mandatory hash binding for all externally referenced artifacts (templates, attachments, originals)
- If encryption is present (non-public contexts only): authenticated decryption (AEAD, hybrid classical + post-quantum as configured) with fail-closed rejection on any decryption or authentication failure
- Fail-closed rejection on any signature, hash-binding, canonicalization, or policy-verification failure

Encryption provides *additional protection where used*, but tamper resistance is fundamentally enforced by signatures and hash binding, not by encryption alone.

Risk 2 — Capsule Forgery or Sender Impersonation

Risk:

An attacker forges a Capsule or impersonates a legitimate sender.

Mitigation / Solution:

- Post-quantum digital signatures bind Capsule content to sender identity
- Sender public keys are verified according to BEAP / WR Code trust rules
- Any signature mismatch results in fail-closed rejection
- For public WR Codes, publisher identity is cryptographically attributable (e.g. registry identity, DNS verification, payment linkage)

Risk 3 — Template Substitution or Semantic Downgrade

Risk:

A malicious actor replaces or alters a template while keeping identifiers or metadata intact.

Mitigation / Solution:

- Capsule embeds the template hash inside the signed canonical Capsule payload
- Receiver recomputes and verifies the hash against the referenced template

- Any mismatch results in immediate fail-closed rejection
- No implicit, registry-based, or version-based trust is permitted

Risk 4 — Registry or Hosting Compromise

Risk:

A public registry, mirror, or CDN serves a modified or malicious template.

Mitigation / Solution:

- Template content is accepted only if its hash matches the Capsule reference
- Registries and mirrors are treated as untrusted transport, not trust anchors
- Optional anchoring (e.g. Solana, IOTA) may provide external immutability evidence
- Compromised hosting cannot bypass local hash verification

Risk 5 — Malicious Public Template Performing Harmful Automation

Risk:

A publicly published template attempts harmful but technically valid automation.

Mitigation / Solution:

- Templates are publicly visible and continuously auditable
- Publisher identity is persistent and attributable
- Pre-publication and on-device static analysis
- Explicit user consent is mandatory before execution
- Capability-bounded execution environment
- Augmented warnings for suspicious scopes, permissions, or targets

Risk 6 — Social Engineering or Consent Manipulation

Risk:

A user is tricked into approving harmful automation.

Mitigation / Solution:

- Consent is the final and mandatory execution gate
- Consent UI MUST display:
 - publisher identity
 - template identifier and hash

- requested actions and execution scope
- High-impact actions require elevated or multi-step consent
- Anomaly detection triggers explicit warnings
- Scan, parse, or verify - execution by protocol definition

Risk 7 — Replay of Previously Valid Capsules

Risk:

An attacker replays a previously valid Capsule to retrigger actions.

Mitigation / Solution:

- Capsule-level unique identifiers / nonces
- Receiver-side replay cache
- Optional timestamp and expiry enforcement
- Optional anchoring of execution log roots for dispute resolution

Risk 8 — Attachment Substitution or Binary Injection

Risk:

An attacker replaces or injects malicious binaries referenced by a Capsule.

Mitigation / Solution:

- Attachments are referenced by cryptographic hash inside the signed Capsule
- Binary content is accepted only if the hash matches
- Attachments are never executed
- Viewing uses rasterized previews where applicable
- Decryption of attachment binaries, if permitted by policy, occurs only outside WRGuard-protected environments
- Decrypted binaries are never directly accessible inside protected environments
- Any mismatch results in fail-closed rejection

Risk 9 — Parser, Decoder, or Rasterization Exploits**Risk:**

Malformed Capsules, templates, or attachments exploit parsing or rendering components.

Mitigation / Solution:

- Strict schema validation
 - Bounded input sizes and deterministic parsing
 - Separation of parsing and rasterization from core verification logic
 - Sandboxed execution for heavy or unsafe processing components
 - Fail-closed behavior on any parse or validation error
-

Risk 10 — Unauthorized Privilege Escalation During Public Execution

Risk:

Publicly executable Capsules gain access to privileged device capabilities.

Mitigation / Solution:

- Public execution is permitted only for WR Codes classified as public
- Public execution does not imply trust or authorization
- Public execution environments enforce:
 - no vault or secret access

- no identity elevation
- no privileged side effects without explicit consent
- All cryptographic verification remains mandatory
- Private and encrypted WR Codes MUST NOT be executable via public execution paths

Risk 11 — Key Compromise on Sender or Receiver Devices

Risk:

Private keys are extracted or misused on compromised endpoints.

Mitigation / Solution:

- Key isolation via OS keystores or hardware-backed storage where available
- Separate keys for public execution vs private or paired contexts
- Endpoint compromise is treated as endpoint trust loss, not protocol failure
- Optional runtime integrity enforcement (e.g. WRGuard-style mechanisms)

Risk 12 — Publisher Identity Spoofing or Look-Alike Attacks

Risk:

An attacker imitates a legitimate publisher to deceive users.

Mitigation / Solution:

- Registry admission rules (DNS verification, payment binding)
- Persistent publisher identifiers
- Publisher identity prominently displayed during consent
- Public auditability and third-party scrutiny of templates

Explicit Residual Risks (Non-Goals)**Risk:**

A fully informed user intentionally approves harmful automation.

Mitigation / Solution:

- Treated as informed consent, not protocol failure
- Transparency, auditability, and warnings reduce likelihood
- The protocol does not override user agency

Security Invariants (Must Always Hold)

- No execution occurs without explicit user consent.
- No template or attachment is admissible unless its hash has been successfully verified.
- Partial verification and downgrade paths are not permitted.
- Trust elevation is permitted only where explicitly defined by policy and supported by verification.
- Admission is conditional upon a matching receiver role (e.g., CTO) and a verified enterprise-grade orchestrator where such requirements are defined.
- Public WR Codes remain unencrypted and publicly auditable at all times.

Proof of Execution (PoE) in the BEAP Ecosystem: A Cryptographically Verifiable Framework for Deterministic, Secure, and Legally Robust Automation

Abstract

The Bidirectional Email Automation Protocol (BEAP) defines a transport-agnostic automation channel designed for secure, verifiable, and reproducible machine-to-machine workflows. At its core lies the

Capsule Protocol, inside a structured cryptographic envelope, containing deterministic execution semantics. The protocol integrates an optional Proof of Execution (PoE) mechanism, which attests that a workflow was executed by an uncompromised orchestrator under validated templates, configurations, and temporal constraints. PoE relies on canonical hashing, hardware-backed runtime attestation, hybrid encryption, and a flexible Time-Critical Execution (TME) model capable of supporting digital, robotic, and deferred workflows. It further incorporates asynchronous verification to avoid execution latency while ensuring full correctness before PoE is finalized. Throughout every stage of the protocol, the receiver retains sovereign control over execution permissions and policy enforcement. Together, these mechanisms create a secure and legally admissible foundation for AI-driven automation across distributed environments.

1. Introduction

Distributed automation requires trust in both the authenticity of instructions and the correctness of their execution. Conventional cryptographic methods protect communication channels but do not verify that a workflow was executed faithfully, deterministically, or within an uncompromised environment. BEAP addresses this gap through the Capsule Protocol, Proof of Execution (PoE), WRCode deterministic templates, secure enclave attestation, and a principled enforcement model where the receiver always determines what is allowed. The framework additionally supports asynchronous verification, ensuring that high-performance or real-time workflows do not incur unnecessary execution delays.

2. The Capsule Protocol

The Capsule Protocol defines a deterministic, cryptographically sealed instruction object that regulates execution semantics, integrity requirements, temporal constraints, and auditability.

2.1 Template Identity and Binding

Each capsule binds to a WRCode template via:

- a unique template identifier and version,
- a canonical template hash (H2a),
- allowed parameter definitions,
- operational constraints.

H2a is computed as:

$H2a = \text{SHA-256}(\text{CanonicalJSON}(\text{TemplateStructure} \parallel \text{Metadata}))$

Canonical serialization ensures identical hashing across implementations.

2.2 Configuration Integrity

Runtime configuration is hashed into H2b:

$H2b = \text{SHA-256}(\text{CanonicalJSON(Configuration)})$

This includes LLM parameters, safety rules, agent behavior specifications, and vector lookup scopes.

2.3 Allowed Execution Modes

Capsules specify permissible execution modes, including manual approval, sandbox-only operation, deterministic-strict workflows, network restrictions, or full automation. These constraints shape how the orchestrator may interpret and execute the capsule.

2.4 Receiver Sovereignty

The receiver retains absolute control over what the orchestrator may execute. Capsules may request data, PoE logs, extended proofs, or elevated permissions, but the receiver's local policy engine determines whether requests are honored, restricted, modified, or denied. Only under mutually agreed automation contracts do capsule requests potentially map to full execution allowances.

2.5 PoE Requirements and Levels

Capsules may request PoE at varying levels:

- minimal (template and configuration verification),
- standard (environment attestation included),

- full (with timing constraints),
- audit-grade (including specific PoE log excerpts).

The receiver decides whether these requests are permitted.

2.6 Time-Critical Execution (TME)

Capsules define:

- a start-after timestamp,
- a finish-before deadline,
- a bounded execution window.

TME prevents pre-execution, delayed execution, or replay scenarios.

2.7 Hash References and Template Chains

Capsules may include references to parent templates or multi-step workflows, optionally encoded via Merkle-style chains to ensure causal integrity across multiple execution steps.

2.8 Key Identifiers

Key identifiers for verifying signatures, orchestrator attestation keys, and secure enclave measurement keys are included, enabling precise cryptographic validation.

2.9 Replay Protection

Capsules incorporate nonces, salts, and globally unique capsule identifiers to ensure one-time semantics and prevent replay attacks.

2.10 State Machine Semantics

The capsule defines deterministic state transitions for sender and receiver, regulating the phases of receipt, validation, execution, verification, PoE generation, and response.

2.11 Data and PoE Log Requests

The capsule may request specific structured data or selected PoE log segments, specified through schema-bound descriptors. These requests are subject to receiver policy enforcement.

2.12 Execution Envelope and Resource Constraints

Capsules may impose resource ceilings, token limits, sandbox restrictions, and compliance requirements. The receiver may override these based on local policy.

2.13 Forensic Determinism

Errors, validation results, and execution traces are serialized deterministically, ensuring reproducibility and auditing consistency.

2.14 Asynchronous Verification

To minimize execution latency, the orchestrator may begin workflow execution immediately while performing verification tasks in parallel. Template re-hashing, configuration verification, and environment integrity checking may occur asynchronously, provided all verification steps are completed before PoE is finalized.

This model is formalized as:

Execution_start → parallel_verification → Execution_end → Verification_complete → PoE_generation

PoE generation is permitted only when verification has completed successfully. If verification fails, the orchestrator enters a validation-failure state and produces a PoE error artifact.

3. Hashing Methodology

PoE relies on nested, canonicalized hashing to ensure deterministic reproducibility across heterogeneous environments.

3.1 Canonical Serialization

All components—templates, configurations, environment attestations, and logs—are serialized using deterministic encoding rules. This ensures consistent byte-level representation for hashing.

3.2 Multi-Layer Hash Composition

PoE integrates four primary hashes:

- H2a: template integrity
- H2b: configuration integrity
- EMH: environment measurement hash
- TME hash: temporal constraint serialization

The final PoE hash is:

$\text{PoE} = \text{SHA-256}($

H2a ||

H2b ||

EMH ||

CanonicalJSON(TME) ||

SHA-256(CanonicalJSON(ExecutionLogMinimal))

)

3.3 Environment Measurement Hash (EMH)

EMH is derived from secure enclave measurements:

EMH = SHA-256(

OrchestratorBinaryHash ||

DependencyHashes ||

EnclaveMeasurement

)

PoE cannot be generated if EMH does not match expected values.

3.4 Hash Chains for Multi-step Workflows

For multi-stage workflows:

H_workflow = SHA-256(PoE_step1 || PoE_step2 || ... || PoE_stepN)

This enables deterministic provenance across extended operations.

4. Capsule Encryption and Decryption

Capsules employ hybrid encryption to ensure confidentiality and integrity.

4.1 Encryption

1. Serialize capsule into canonical JSON.
2. Generate a symmetric session key K_{session} using a secure RNG.
3. Encrypt capsule bytes using AES-256-GCM:

$\text{Ciphertext} = \text{AES-256-GCM}(K_{\text{session}}, \text{CapsuleBytes})$

4. Encrypt the session key using the receiver's public key:

$\text{EncryptedKey} = \text{RSA-OAEP}(\text{receiver_public_key}, K_{\text{session}})$

5. Construct the encrypted capsule envelope with ciphertext, encrypted key, nonce, and authentication tag.

4.2 Decryption

1. Decrypt K_{session} using the receiver's private key.
2. Decrypt the ciphertext using AES-256-GCM.

3. Validate the authentication tag.
4. Parse and canonicalize the capsule.
5. Apply receiver policy before any execution is permitted.

4.3 Signing

Signing is distinct from encryption. Capsules are signed by the sender to prove origin and prevent tampering. Unsigned or invalidly signed capsules are rejected.

5. Proof of Execution (PoE)

PoE attests that a workflow was executed faithfully, under correct templates, configurations, and environment conditions.

5.1 PoE Components

PoE integrates:

- H2a (template integrity),
- H2b (configuration integrity),
- EMH (secure enclave attestation),

- TME constraints,
- minimal execution logs.

PoE is generated only after all verification tasks have completed successfully.

5.2 Secure Enclave Attestation

The orchestrator may operate within a trusted execution environment (such as TPM, SGX, TrustZone, or Secure Enclave) that measures binary integrity, module identity, and runtime stability. If measurements do not match expected values, PoE generation, rasterization of original binaries and encryption of capsules is prohibited.

6. Time-Critical Execution (TME)

TME enforces temporal correctness.

6.1 Digital Workflows

Digital tasks typically execute within narrow TME windows, enabling immediate PoE validation.

6.2 Extended TME for Robotics and Industrial Automation

Physical workflows may require extended or variable time. BEAP supports this through a two-tier model:

- Tier 1: Pre-Execution PoE under strict time constraints
- Tier 2: Physical execution window with extended duration

Completion yields a Final PoE bound cryptographically to the Pre-Execution PoE.

6.3 Deferred Execution

Deferred or scheduled tasks provide deadlines and re-attestation requirements to preserve integrity across long-running workflows.

7. Receiver Sovereignty and Mutual Agreements

The receiver always retains ultimate authority over what may be executed, whether PoE logs are returned, and how capsule instructions are interpreted. Only under mutually defined automation agreements do capsule requests and recipient policy align fully.

8. Legal and Evidentiary Strength

PoE satisfies criteria for admissible technical evidence:

- authenticity via signatures and secure enclaves,

- integrity through canonical hashing,
- non-repudiation via hardware-bound attestation,
- temporal correctness via TME,
- reproducibility through deterministic serialization,
- chain-of-custody via state machine logs.

9. Conclusion

BEAP's Capsule Protocol, combined with WRCode templates, secure enclave-anchored PoE, canonical hashing, hybrid encryption, and receiver-centric policy enforcement, forms a comprehensive execution integrity framework for distributed automation. Through asynchronous verification, PoE imposes no latency overhead on time-critical workloads while still ensuring full correctness before proof generation.

Optional Execution Controls and Role Management in BEAP Capsules

The BEAP Capsule format is designed to support a wide range of automation scenarios—from simple, single-step interactions to complex, regulated, multi-party workflows. To achieve this flexibility without overcomplicating minimal use cases, BEAP introduces a fully optional control layer:

approval chains, resource constraints, execution policies, provenance declarations, and more can be defined only when needed. If these fields are omitted, the capsule executes under default sandboxed policies and receiver-side control.

This opt-in design makes BEAP suitable for both lightweight automations and enterprise-grade governance.

Role-Based Approval and Execution Authorization

Capsules may optionally declare role-based approval requirements. These allow the orchestrator to enforce multi-level review processes involving employees, managers, finance officers, or external auditors. If no required_approvers are defined, the orchestrator will default to the receiver's local policy (e.g. manual or auto-approve).

Example:

```
"required_approvers": [  
    { "role": "employee" },  
    { "role": "supervisor" },  
    { "role": "cfo", "conditions": ["invoice_amount > 10000"] }  
]
```

Each role is resolved via the receiver's identity system or WRAccount mapping. Approvals can be time-limited, delegated, or escalated if inactive.

Optional Delegation and Signing Logic

Capsules may embed delegation proofs that authorize a substitute actor to approve or execute within defined scopes and timeframes:

```
"delegation": {  
    "from": "cfo@org.com",  
    "to": "deputy@org.com",  
    "valid_until": "2026-01-01",  
    "scope": ["finance.invoice.approval"]  
}
```

Cryptographic signatures from approvers or delegates are enforced only if signatures_required is present.

Optional Execution Constraints and Policies

All enforcement mechanisms in BEAP are declarative and optional. When present, they are enforced strictly and verifiably. When omitted, the capsule falls back to orchestrator-defined defaults.

| Field | Function (if present) |
|-----------------------|--|
| approval_deadline | Defines a timebox for decision-making |
| on_error | Specifies retry or fallback behavior |
| expected_state | Declares required before/after states for determinism |
| resource_requirements | Filters eligible execution environments |
| billing | Supplies quota, licensing, or cost metadata |
| regional_context | Triggers locale-aware routing or legal restrictions |
| input_provenance | Ensures auditability of embedded or linked content |
| replay_policy | Enforces one-time or reusable execution semantics |
| classification | Tags the capsule for dashboard sorting, analytics, or queueing |

| Field | Function (if present) |
|--------------|---|
| view_filters | Allows selective visibility per role or interface component |

Each of these fields is optional. If defined, they become binding; if not, execution proceeds under local defaults.

Custom Logic and Adapter Integration

BEAP capsules may define or reference custom runtime logic, enabling organizations to adapt execution policies, approval rules, or business processes to their internal workflows.

Examples include:

- Custom precondition evaluators (e.g. field presence, LLM evaluation, external API responses).
- Adapter modules for CRM, ERP, or HR systems.
- Custom vectorstore or semantic lookup integration.

Extension example:

```
"custom_logic": {  
    "precondition_handler": "module://company.crm.check_credit_limit",
```

```
"approval_handler": "module://finance.workflow.dynamic_routing"  
}
```

Custom modules must be signed and permitted by local policy before execution. This ensures safe extensibility without compromising the deterministic behavior of core capsules.

Execution Defaults and Fallback Behavior

If no constraints or policies are declared, the BEAP orchestrator will:

- Validate the capsule structure and signature.
- Apply local receiver policy (manual, semi-auto, or full-auto).
- Execute the request using the safest compatible defaults.

This ensures maximum compatibility while still enabling fine-grained enforcement when security or regulatory needs require it.

Conclusion

The BEAP Capsule framework is flexible by design:

- Nothing is mandatory except the core envelope structure.

- Everything else is optional—but strictly enforced if defined.
- Custom logic can be added modularly for enterprise or domain-specific workflows.
- Role management, delegation, signing, resource limits, and approval deadlines provide enterprise-grade control without burdening minimal use cases.

This allows BEAP to scale from lightweight AI task routing to full-scale, trust-anchored inter-organizational automation—all using a single verifiable capsule model.

Transport-Agnostic Machine-to-Machine Automation and Execution Modes in BEAP

While BEAP Capsules can always be exchanged and processed manually (for example, via file upload or human-initiated mail handling), the protocol is primarily designed to enable fully automated, machine-to-machine (M2M) communication across heterogeneous transport channels. In this mode, Capsules function as structured, verifiable API calls between orchestrators—without requiring a shared HTTP API, VPN, or tightly coupled infrastructure.

This section describes how BEAP enables transport-agnostic M2M automation, how Channel Adapters bridge different communication environments, and how execution modes and approval policies provide safe, controllable automation—from fully human-in-the-loop up to fully machine-driven workflows, with a partially human-in-the-loop mode in between.

1. BEAP as a Virtual API Layer

At the M2M level, a BEAP Capsule can be viewed as a self-describing remote procedure call:

- The intent acts like an API route or method (e.g., invoice.submit, status.query, robot.move_segment).
- The payload carries structured data, parameters, and references.
- The policy block declares how the receiver is allowed to execute and respond.
- The Capsule state machine (REQUEST → RECEIPT → RESPONSE → optional Proof of Execution) defines the interaction lifecycle.

Instead of exposing a public HTTP endpoint, organizations expose a BEAP Inbox—any channel where Capsules can be delivered and extracted. Two orchestrators can then interact as follows:

1. The sender constructs a REQUEST Capsule with intent, payload, and policy.
2. The Capsule is transmitted over any available transport (email, messaging, file, IPC, offline QR handover).
3. The receiver's orchestrator validates, interprets, and optionally executes the Capsule.

4. The receiver responds with a RECEIPT and, if appropriate, a RESPONSE Capsule and PoE artifact.
5. The sender validates the returned Capsules and updates its own state machine and audit log.

The underlying transport is irrelevant as long as both sides can reliably send and receive Capsule bytes.

2. Channel Adapters: Bridging Transports Without Changing the Protocol

To keep the Capsule format stable and interoperable, BEAP introduces the concept of Channel Adapters. A Channel Adapter is a local component that:

- Receives raw messages from a specific transport (e.g., email, WhatsApp Web, Signal Desktop, HTTP, message queues).
- Extracts potential Capsule payloads (attachments, structured blocks, tagged segments).
- Hands them to the BEAP orchestrator as pure data (without rendering or trusting surrounding content).
- Encodes and transmits outgoing Capsules back over the same or a different transport.

Common examples include:

- Email Adapter

Capsules are carried as application/beap+json attachments or structured body segments. The adapter uses IMAP/POP/API to fetch messages, isolates the Capsule bytes, and passes them into the orchestration pipeline.

- Messaging Adapter (API-based)

For platforms with official APIs or webhooks, Capsules can be sent and received as documents or structured payloads. The adapter maps these messages to Capsules and back.

- Browser/Web Adapter (DOM-driven)

For platforms with web interfaces but no official API (e.g., certain messengers), Capsules can be extracted from the DOM within a "Master Tab".

The adapter:

- reads the DOM, isolates Capsule segments, and forwards them as data;
- injects replies or attachments into the UI when required, optionally with user confirmation.
- File/IPC/Offline Adapters

Capsules can be stored in shared directories, exchanged via SFTP, or moved on portable media. IPC adapters route Capsules between local processes or devices.

From the Capsule's perspective, all of these appear as the same structure. The Capsule format never changes; only the adapter implementation does.

3. Request, Receipt, Response, and Optional Confirmation-of-Confirmation

To support legally robust, non-repudiable automation, BEAP extends the basic REQUEST/RESPONSE pattern with an explicit Receipt phase and an optional confirmation-of-confirmation.

3.1 Request Capsule

A REQUEST Capsule initiates a workflow:

- capsule_type: "REQUEST"
- capsule_id (globally unique)
- intent, payload, policy
- optional PoE requirements
- optional time windows and Time-Critical Execution (TME) constraints

3.2 Receipt Capsule

Upon receiving and validating a REQUEST, the orchestrator may generate a RECEIPT Capsule:

- capsule_type: "RECEIPT"
- in_reply_to: <capsule_id of REQUEST>
- status fields (received, accepted, rejected, etc.)
- timestamps and optional reason codes

The RECEIPT is cryptographically signed and binds the receiver to the statement that the Capsule was received and interpreted at a specific time under a specific identity. With a proper key management and attestation chain, this receipt is no longer plausibly deniable.

3.3 Response Capsule

After executing the requested workflow (or rejecting it), the orchestrator may send a RESPONSE Capsule:

- capsule_type: "RESPONSE"
- in_reply_to (REQUEST or RECEIPT, depending on configuration)
- structured result, error information, or status update
- optional PoE hash and references

3.4 Optional Receipt-Acknowledgement Capsule

For highly regulated environments or sensitive transactions, senders and receivers may opt into a stronger pattern:

- REQUEST → RECEIPT → RESPONSE → RECEIPT_ACK

The RECEIPT_ACK confirms that the sender:

- has received the RECEIPT (and/or RESPONSE),
- has integrated it into their own state machine and PoE log.

This symmetric pattern reduces disputes such as “the acknowledgement never arrived” on either side. It is optional and can be requested in the Capsule policy for specific workflows, partners, or risk classes.

4. Execution Modes: Human Control and Headless Automation

To cover simple and complex environments without forcing unnecessary complexity, BEAP defines three execution modes, enforced locally by the receiver’s orchestrator:

1. Fully human-in-the-loop

2. Partially human-in-the-loop (automatic responses, gated execution)
3. Fully machine-driven

Execution mode is never imposed unilaterally by the sender.

The Capsule may request a mode, but the receiver's local policy decides whether to:

- honor the request,
- downgrade to a safer mode,
- or reject the Capsule entirely.

4.1 Fully human-in-the-loop

- Capsules are validated but never executed automatically.
- The orchestrator:
 - verifies signatures, integrity, and policies;
 - presents a human-readable summary of the request;
 - waits for explicit user approval or rejection.
- Responses and PoE are only generated after explicit human confirmation.

- This mode is appropriate for initial deployments, highly sensitive actions, or environments without established trust.

4.2 Partially human-in-the-loop (assisted mode)

In partially human-in-the-loop mode, responses and orchestration decisions can be prepared automatically, while actual execution remains gated:

- Parsing, validation, policy evaluation, and draft response generation run fully automatic.
- The orchestrator may:
 - generate a RECEIPT automatically,
 - prepare a RESPONSE Capsule as a draft,
 - compute all required PoE inputs (hashes, environment checks, TME evaluation),
 - pre-assemble the execution plan for downstream systems or robots.
- High-impact actions, state changes, or external effects (e.g., payments, contractual changes, robotic movements) are blocked until required approvers confirm.
- Formally:
 - automated responses are allowed,

- actual execution must be confirmed by one or more human roles.

This mode combines speed and machine assistance with a hard human gate for irreversible or high-stakes actions.

4.3 Fully machine-driven

In fully machine-driven mode, Capsules that pass validation and policy checks may be executed end-to-end without any human involvement:

- The orchestrator:
 - validates the Capsule,
 - executes the workflow automatically,
 - generates RECEIPT, RESPONSE, and PoE artifacts,
 - updates local logs and state machines.
- This mode is intended for:
 - recurring workflows with strong contractual agreements,
 - low-risk operational tasks,

- high-frequency interactions where manual gating is impractical or impossible.

Even in this mode, execution remains bound by Time-Critical Execution constraints, PoE requirements, and local policy. The receiver can always refuse, downgrade, or sandbox a Capsule if conditions are not met.

5. Role-Based Policies and Machine-Only Communication

Machine-to-machine BEAP flows often require human involvement only at defined decision points. Optional, role-based mechanisms in the Capsule format make this explicit:

- required_approvers declares which roles must approve execution or a specific step (e.g., employee, supervisor, CFO).
- Delegation rules allow decision-makers to authorize delegates for defined scopes and time windows.
- Execution constraints (amount thresholds, domain restrictions, environment requirements) can be encoded directly in the Capsule.

For purely machine-driven pipelines—such as two backend systems exchanging status updates, telemetry, or low-risk synchronization tasks—Capsules can omit all role and approval fields. In such cases, execution reverts to:

- the receiver's global policy for that sender or WRAccount,
- the configured execution mode (often fully machine-driven for low-risk intents),
- the existing Proof of Execution and Time-Critical Execution constraints.

Thus, BEAP supports the full spectrum:

- fully human-in-the-loop,
- partially human-in-the-loop (automatic responses, gated execution),
- fully machine-driven,

without changing the core Capsule structure.

6. Headless and Semi-Headless Operation over Web Interfaces

Even when a platform exposes only a web interface (for example, desktop or browser-based messengers), BEAP can still operate in assisted or fully automated modes through the Browser/Web Adapter:

1. Messages are captured from the DOM or Web API without rendering Capsules.
2. Capsules are isolated and validated as pure data.
3. Depending on execution mode:
 - Fully human-in-the-loop: the user is shown a safe, overlay-based summary and must decide.
 - Partially human-in-the-loop: the orchestrator prepares responses automatically while gating execution behind explicit approvals.
 - Fully machine-driven: the orchestrator injects responses or triggers follow-up Capsules directly, subject to local policy and PoE/TME constraints.

This allows organizations to use existing consumer or enterprise messaging platforms as automation backbones without compromising security, even when the original UI was never designed for machine-to-machine protocols.

7. Non-Repudiation, Auditability, and Legal Robustness

By combining:

- signed Capsules,

- explicit REQUEST / RECEIPT / RESPONSE / optional RECEIPT_ACK,
- Proof of Execution (PoE) anchored in secure enclaves and WRCode templates,
- deterministic hashing across templates, configuration, environment measurements, TME constraints, and minimal execution logs,

BEAP establishes a machine-verifiable audit trail:

- who requested which action,
- under which template and configuration,
- in which environment and time window,
- what was executed and with what outcome,
- which approvals and confirmations were given.

This record is independent of the underlying channel. Whether Capsules travelled via email, messaging, shared folders, or QR-based offline exchange, the Capsules and PoE artifacts themselves provide a consistent, cryptographically bound proof chain that can be evaluated in both technical and legal contexts.

In summary, BEAP extends beyond “automation over email” into a general, transport-agnostic automation fabric. Capsules become the lingua franca of inter-organizational machine communication: self-describing, verifiable, and enforceable—whether processed fully human-in-the-loop, partially human-in-the-loop, or fully machine-driven. If machine learning or reinforcement learning is enabled in the adapter layer, all inferred mappings and execution decisions remain advisory only and require explicit human consent before activation; adapter execution itself stays fully deterministic.

The WRVault: Dual-Vault Architecture for Encrypted, Local, Zero-Trust Execution

The WRVault is the foundational security substrate of the WRCode ecosystem. It provides encrypted, deterministic, and fully local execution context for Capsules, Mini-Apps, agents, and Proof-of-Execution (PoE) artifacts. Unlike centralized automation systems, the WRVault ensures that all execution state remains on the user’s device, encrypted at rest, and accessible only under strict policy and user-controlled conditions. This creates a zero-trust automation environment where neither remote senders nor automation logic can bypass local authority.

To support both ergonomics and regulated high-assurance workflows, WRCode implements a dual-vault design consisting of a standard login-bound vault (WRVault-A) and multiple optional

password-protected high-security vaults (WRVault-B instances). Each vault operates with independent encryption, access rules, and lifecycle governance.

1. WRVault-A — Standard Encrypted Execution Vault (Login-Bound)

WRVault-A serves as the default operational vault for routine automation. It stores:

- Mini-App state and deterministic UI transitions
- contextual embeddings and semantic lookups
- agent decisions and execution metadata
- Capsule state and normalized instructions
- standard PoE components and replay-relevant artifacts
- non-sensitive local logs required for reproducibility

1.1 Encryption and Access Control

WRVault-A is encrypted using a device-bound key sealed to the WRCode account.

Upon successful WRCode login, the orchestrator automatically unseals the vault under strict local policy.

Even though access is ergonomic, several boundaries remain:

- By default, no data can ever leave the device under any circumstances.
Data exchange becomes possible only when both parties have an established handshake and mutually aligned policies explicitly permitting that exchange. Without this agreement, all data remains strictly local.
The receiver is always the ruling party: a capsule cannot escalate privileges, cannot override local policy, and cannot force any form of data release or execution.
- Agents and Mini-Apps cannot bypass access boundaries.
- No vault content is ever transmitted externally without explicit user permission.

All outbound requests, whether triggered by Mini-Apps, agents, Capsule responses, or PoE generation, pass through a mandatory outbound security-policy engine. This engine enforces that data never leaves the device unless the user or organization explicitly authorizes it.

1.2 Local Data Boundary

By default, WRVault-A enforces:

- no background syncing
- no implicit sharing

- no silent export
- no indirect data access through Capsule logic

This preserves user sovereignty in all day-to-day automation scenarios.

2. WRVault-B — Password-Protected High-Security Vaults

For sensitive, regulated, or legally binding workflows, the system supports WRVault-B: one or more high-security vaults with independent keys and strict isolation.

Each WRVault-B instance stores:

- sensitive Mini-App state and confidential inputs
- PII-bound or regulatory context fragments
- high-assurance PoE trails and timing evidence
- legal-grade audit logs and chain-of-custody records
- data requiring explicit human authorization

2.1 Dual-Factor Access Model

Access to any WRVault-B requires both:

1. a valid WRCode login session (identity and authentication), and
2. an additional high-security password (user-gated approval).

This ensures:

- the orchestrator cannot silently access high-security data,
- Capsules cannot unlock or read high-security vaults,
- Mini-Apps and agents cannot escalate privileges (the orchestrator of the receiving party is always also in operational mode the only source of truth, settings are just stricter per default),
- the user remains the final authority over all high-security operations.

This rule is enforcement-critical: *Capsule instructions cannot unlock vaults by design* .

2.2 High Security Mode

When the user activates High Security Mode:

- high-sensitivity logs are redirected into a WRVault-B instance,
- stricter PoE logic and TME rules apply,

- outbound data transmission becomes user-gated only,
- Mini-Apps may require stepwise user approval,
- all high-security evidence remains local and password-gated.

Switching between operational and high-security modes never merges or exposes historical data across vaults. Each vault preserves its own encryption key and access lifecycle.

3. Multi-Vault Support and Role Isolation

A unique property of the architecture is that multiple WRVault-B instances may coexist on the same device. Each instance functions as an independent high-security compartment with:

- its own encryption key
- its own password
- its own role and access permissions
- its own retention and audit policies

This enables granular separation between:

- different employees
- departments or business units
- regulatory domains
- personal vs. organizational data
- contractual or legal contexts

Because each WRVault-B is cryptographically and procedurally isolated, no user can access another user's high-security data unless explicitly granted via organizational policy. This supports multi-user machines, shared workstations, and enterprise deployments with strict role-based governance.

4. Internal Structure of the Vaults

Both WRVault-A and WRVault-B share a similar structural design, but with differing access rules:

4.1 Capsule State Compartment

Stores canonicalized Capsule data:

- intent

- routing
- Template/Configuration hashes (H2a, H2b)
- TME constraints
- execution policies

4.2 Deterministic Execution Log Chain

A tamper-evident log chain built from timestamped, canonical entries:

$$\text{log}[n] = \text{SHA-256}(\text{log}[n-1] \parallel \text{entry}[n] \parallel \text{timestamp})$$

WRVault-B logs form part of high-assurance, regulator-grade PoE trails.

4.3 Mini-App State Containers

Each Mini-App is sandboxed in its own subcontainer:

- deterministic UI evolution
- agent decision graphs
- semantic memory used during execution
- security policies and constraint evaluations

4.4 Sensitive Context Compartment

Stores:

- derived embeddings
- confidential documents or state fragments
- user-supplied sensitive information
- high-assurance context for PoE

This compartment is never exposed directly—only hashed commitments appear in PoE artifacts by default, ensuring the user always remains in full control over what information is revealed.

5. Vault Integration with Proof of Execution (PoE)

PoE derives exclusively from Vault entries, never from untrusted message bodies or remote sources.

WRVault-A PoE

- operational-grade
- reproducible execution traces

- deterministic logs
- standard TME validation

WRVault-B PoE

- regulator-grade
- sealed evidence chains
- high-integrity TME records
- user-approved provenance
- legally defensible execution history

A PoE derived from WRVault-B is cryptographically linked to the moment the user unlocked the vault, anchoring the proof to explicit human consent.

6. Security Architecture

The dual-vault model enforces:

- User sovereignty: All high-security access requires explicit approval.

- Zero-trust execution: Capsules and agents cannot access vaults directly.
- Strict local boundaries: Data never leaves the device without explicit permission.
- Tamper-evident logs: All evidence is canonicalized and hash-linked.
- Role isolation: Multiple WRVault-B vaults enforce compartmentalization.
- Defense in depth: High-assurance workflows require two independent factors.
- Regulatory readiness: PoE and state isolation support legal-grade evidence requirements.

BEAP Capsule Generation in the WRCode Ecosystem

BEAP Capsules are the core transport unit for structured automation in the WRCode ecosystem. They are not handwritten code, but deterministic, cryptographically sealed exports of one or more WRCode sessions. Capsules encapsulate Agents, Agent Boxes, Mini-Apps, Augmented Overlays, contextual information, execution constraints, and policy rules—forming a portable, zero-trust automation contract.

A Capsule defines what is intended to happen and under which verified constraints, without transmitting executable code.

1. Capsule as Deterministic Export of One or More Sessions

A *session* is a coherent execution environment containing:

- Agents and collaboration rules
- Agent Boxes and mid-tier logic
- Mini-Apps and their UI state
- Overlay configuration and DOM interaction rules
- contextual inputs
- execution constraints and policy state

When creating a Capsule, the user may export:

- a single session, representing a standalone workflow

or

- multiple sessions into the same Capsule, allowing several tasks to travel together as one cryptographically unified automation unit.

Multi-session Capsules preserve internal isolation between sessions while sharing a unified signature, policy envelope, and PoE trace.

1.1 Multi-Session Capsules

Multi-session export is flexible and not limited to long workflows. It is used when:

- the sender wants to transmit several independent tasks in one Capsule
- multiple Agents or Mini-Apps must run in parallel
- contextual or departmental separation must be preserved
- several stages of automation should be delivered together
- batching improves operational efficiency

Each session is exported independently, then merged into a single Capsule under deterministic rules:

- separate context blocks
- separate policy sections if needed
- unified Capsule metadata
- shared cryptographic binding

This allows downstream orchestrators to execute sessions sequentially, independently, or in parallel, depending on their policies.

1.2 Sub-Capsules for Delegation (Multiple Independent Capsules)

Sub-capsules are not embedded inside a parent Capsule.

Instead, during the creation process, the user may generate multiple independent Capsules in a single step.

This allows:

- delegation of sub-tasks
- distribution of responsibilities
- splitting complex workflows into separate recipients
- fine-grained routing across departments or organizations
- partial automation at multiple endpoints

Each sub-capsule:

- is a standalone BEAP Capsule
- carries its own session(s), context rules, and policy

- has its own signature and lifecycle
- can be sent to the same receiver or entirely different recipients

The Capsule Creator Wizard ensures:

- consistent metadata
- deterministic structure
- explicit routing rules
- clear audit separation

Sub-capsules exist purely for ease of delegation, not for creating hierarchical Capsule structures.

2. Context Handling: Text-Based, Local Embedding, Optional Large-Context Resolution

BEAP Capsules never include precomputed embeddings.

Instead, they carry context primarily as text.

2.1 Default: Local Embedding on the Receiver Side

When a Capsule is received:

- embeddings are computed only locally

- vector representations are stored in the receiver's vectorstore or Vault
- embeddings are never transmitted

This ensures:

- privacy
- consistent behavior across models and versions
- reproducible PoE results
- compatibility with offline processing

2.2 Optional External Context Resolution

When context is too large to embed directly:

- WRCode.org-hosted context bundles may be referenced
- local files or enterprise registries may be referenced
- Vault compartments may be referenced via hash identifiers

All external context resolution is optional and follows strict policy rules.

3. Capsule Creator Wizard

A progressive, user-friendly Wizard enables safe Capsule creation.

3.1 Simple Mode — Minimal, Safe Capsules

For non-technical users:

- select Mini-App or Agent
- add optional text context
- choose simple behavior

The Wizard generates a Capsule with:

- core intent
- default policy
- minimal context rules
- signed and canonicalized metadata

3.2 Intermediate Mode — Operational Automation

For power users:

- Agent Box selection and tuning

- multi-agent orchestration
- context handling strategy
- TME constraints
- approval chains
- Mini-App Tier 1/2/3 optimization
- fallback logic
- outbound policy

Real-time validation keeps Capsules safe and schema-compliant.

3.3 Advanced Mode — Enterprise and Regulatory Workflows

For experts and enterprise integrations:

- multi-session Capsule construction
- creation of multiple independent sub-capsules
- state machine configuration
- secure enclave and attestation preferences

- legal classification and compliance metadata
- resource ceilings and sandbox policy
- multi-department routing
- cross-organizational automation chains

Capsules created in this mode support enterprise-grade, cross-domain workflows.

4. Deterministic Serialization and Cryptographic Sealing

All Capsules are built through a deterministic pipeline:

1. export selected sessions
2. canonicalize structure
3. compute H2a (template hash)
4. compute H2b (configuration hash)
5. bind policy and constraints
6. attach TME windows
7. generate or group sub-capsules if selected

8. apply optional encryption
9. sign each Capsule
10. register Capsule hashes in Vault-A or Vault-B

Determinism ensures:

- identical behavior across orchestrators
- reproducible PoE verification
- consistent legal interpretation

5. Zero-Trust Constraints

Capsules operate under strict zero-trust rules:

- no executable code is ever embedded
- Capsules cannot unlock Vaults
- Capsules cannot override local security policy
- Capsules cannot perform privilege escalation
- Capsules cannot access data outside allowed context

- local embedding rules cannot be overridden
- outbound communication cannot occur without explicit user policy

Capsules define *intent*, not authority.

6. Integration with Vault and PoE

Capsule generation ties directly into Vault and PoE:

- Capsule structure is anchored in Vault metadata
- multi-session Capsules produce clear, auditable traces
- PoE binds execution to verified Capsule semantics
- context is stored locally in Vault compartments
- high-security Capsules use WRVault-B metadata for provenance

This forms a complete trust chain:

Capsule → Execution → Vault → PoE

ensuring correctness, verifiability, and audit readiness.

7. Summary

The BEAP Capsule Generation system delivers:

- deterministic export of one or more sessions
- the ability to generate multiple independent Capsules for delegation
- local embedding for privacy and reproducibility
- optional WRCode.org context references
- a powerful multi-mode Capsule Creator Wizard
- strict zero-trust boundaries
- deep integration with Vault and PoE for legal robustness
- enterprise-ready automation across email, messengers, or offline channels

Capsules become portable automation contracts that maintain trust, sovereignty, and security across distributed environments. If machine learning or reinforcement learning is used in the Capsule Builder, it is limited to pre-consent assistance for suggesting schema fields or type pairs, while final Capsule structure and all stamped data are created only after explicit human confirmation and remain fully deterministic.

BEAP Capsule Generation in the WRCode Ecosystem

The BEAP Capsule serves as a cryptographically verifiable automation contract within the WRCode ecosystem. Instead of transmitting executable code, a Capsule is a deterministic export of one or more WRCode sessions, enriched with additional metadata needed for cross-organizational workflow execution.

The Capsule describes *what* should happen, *under which constraints*, in a way that is reproducible, auditable, and independent of the transport medium.

Sessions supply the operational content; the Capsule Builder supplies the missing structural and policy elements.

1. Capsules as Deterministic Exports of One or More Sessions

In WRCode, a session represents a complete execution environment, capturing everything that occurred during a workflow:

- Agents and their behavior models
- Agent Boxes and configuration

- Mini-Apps and their UI state
- Augmented Overlay configuration
- contextual text and metadata
- session-specific constraints
- interaction state

A Capsule does not redefine these elements.

Instead, the user selects one or multiple sessions from the session history, and the Capsule is built around those sessions.

Single-session Capsules

Used for isolated workflows or focused tasks.

Multi-session Capsules

Used to transmit multiple workflows or tasks in one cryptographic envelope.

Each session remains internally isolated and independently interpretable by the receiver.

1.1 Multi-Session Capsules

Multi-session Capsules are used when the sender wants to bundle multiple tasks together.

This is not limited to multi-stage workflows — common use cases include:

- batching several independent requests
- sending multiple Mini-Apps at once
- initializing several agent configurations in a single delivery
- preparing parallel workstreams
- grouping related tasks for coordination

Each session is preserved with its own logic, while the Capsule provides a shared transport envelope, signature, and metadata block.

The receiver processes each session independently under its own local policy.

1.2 Sub-Capsules for Delegation (Independent Capsules Created Together)

Sub-capsules are not embedded inside a parent Capsule.

Instead, the Capsule Builder can generate multiple, separate Capsules during a single creation process.

This is useful for:

- delegating tasks to different recipients
- distributing responsibilities across departments
- producing multiple Capsules that share context or configuration
- generating multiple approval chains simultaneously

Each sub-capsule:

- is a standalone BEAP Capsule
- has its own signature, constraints, and policy
- can be transmitted independently
- has its own execution lifecycle and PoE trace

This allows workflows to be decomposed without creating complex nested structures.

2. Context Handling: Text-Based, Local Embedding, Optional External Sources

BEAP Capsules do not contain embeddings.

Context is included primarily as text, ensuring privacy, determinism, and clarity.

2.1 Default: Local Embedding on Receiver Side

When a Capsule is received:

- embeddings are computed exclusively on the receiver's device
- embedding vectors reside only in the receiver's vectorstore or Vault
- the sender never transfers vector data

This enables privacy, consistency, and reproducibility across different orchestrators.

2.2 Optional External Context Resolution

For large contexts, a Capsule may reference:

- WRCode.org context bundles or vectorstores (optional feature)
- enterprise registries
- local files present on the receiver's machine
- Vault compartments (A or B) by reference

These references are policy-bound and never processed automatically unless explicitly allowed.

Local embedding remains the canonical method across all implementations.

3. Capsule Creator Wizard — One Unified Builder, Expandable On Demand

There are no separate “simple,” “intermediate,” or “advanced” modes.

The Capsule Builder is a single, unified interface.

It starts lean and expands only when users enable additional fields.

The workflow is:

1. Select sessions from session history
2. The Builder imports the full session content:
 - o agents
 - o agent boxes
 - o mini-app state
 - o augmented overlay settings

- context
 - session metadata
3. The Builder asks only for the missing information required to transform these sessions into a complete Capsule
 4. Users can optionally enable advanced configuration areas as needed

This maintains usability while enabling highly granular control.

3.1 Baseline Configuration — Session-Based, Minimal, Safe

By default, the Capsule Builder asks only for:

- which session(s) to export
- the desired execution behavior (manual, partial, automatic)
- a short description or label (optional)

All other data is inherited directly from the selected sessions.

The resulting Capsule includes:

- deterministic session export
- safe default policy
- minimal configuration
- digital signature

No advanced fields are displayed unless explicitly enabled.

3.2 Optional Operational Extensions — Add Missing Execution Details

Users may enable fields that extend a session with behavior not contained inside the session itself:

- data requests (what the receiver should return)
- execution requests (what the receiver should perform)
- additional context resolution rules
- outbound communication restrictions
- basic approval logic
- fallback / retry behavior
- TME windows (start-after, latest-accepted-before)

Each block is isolated, optional, and validated in real time.

The Capsule Builder remains clean and manageable.

3.3 Optional Enterprise & Regulatory Extensions

For professional, multi-department, or legally relevant workflows, the Builder can reveal advanced fields such as:

- multi-role approval requirements
- conditional routing logic
- delegation (generating multiple Capsules)
- secure enclave / attestation requirements
- compliance and classification metadata
- custom business logic handlers
- resource ceilings and sandbox definitions
- Vault selection (A, B, or multiple B vaults)
- PoE requirements

- cross-organizational workflow mapping
- state machine specifications

Again, all of these features are *optional* and appear only when enabled.

The Capsule Builder stays consistent regardless of complexity.

3.4 Lean Interface, Maximum Flexibility

The Builder is intentionally minimal by default.

Advanced configurations—policies, constraints, data requests, automation mode definitions, approval roles, custom business logic—can be added only when needed.

This ensures:

- simplicity for beginners
- precision for experts
- consistent Capsule structure
- clear auditability

The system scales naturally from simple tasks to enterprise workflows.

4. Deterministic Serialization and Cryptographic Sealing

Every Capsule is constructed through the same deterministic process:

1. export selected sessions
2. canonicalize
3. compute H2a (template hash)
4. compute H2b (configuration hash)
5. integrate policy, permissions, constraints
6. apply TME rules
7. generate any sub-capsules
8. optionally encrypt
9. digitally sign
10. register Capsule metadata in Vault-A or Vault-B

This ensures cross-implementation reproducibility, legal robustness, and stable PoE verification.

5. Zero-Trust Enforcement

Capsules obey strict zero-trust constraints:

- no executable code allowed
- Capsules cannot unlock or access Vaults
- Capsules cannot override local policy
- Capsules cannot access external data without explicit permission
- Capsules cannot cause privilege escalation
- Capsules cannot bypass TME, PoE, or sandbox rules

A Capsule expresses intent, never authority.

6. Integration with Vault and PoE

Capsule Generation is tightly integrated with Vault and PoE:

- H2a/H2b are stored for reproducibility
- multi-session Capsules map to multiple Vault contexts
- data requests map to Vault compartments for local storage
- high-security workflows store metadata in WRVault-B

- PoE binds execution to the validated Capsule structure

This forms a provable chain-of-custody:

Capsule → Execution → Vault → PoE

ensuring reliable, audit-ready automation.

7. Summary

The BEAP Capsule Generation system provides:

- deterministic export of one or many WRCode sessions
- optional generation of multiple independent Capsules for delegation
- context-first workflows with local embedding
- optional WRCode.org vectorstore integration
- one unified Capsule Builder with optional operational and enterprise extensions
- strict zero-trust architecture
- seamless integration with Vault and PoE

BEAP Capsules are transport-agnostic, audit-ready automation contracts that maintain full user sovereignty while enabling scalable, professional, and legally robust automation across distributed systems.

Extended Operational Architecture of the BEAP and WRCode Ecosystem

Revocation, Execution Semantics, Abort/Undo Behavior, Concurrency, Cryptographic Lifecycle, and Verification Pathways

The BEAP Capsule framework and the WRCode orchestrator define a structured, deterministic environment for distributed automation across messaging channels, email, and P2P transports.

Beyond Capsules, Mini-Apps, Vaults, and PoE, the system incorporates additional operational mechanics: revocation, session lifecycle management, idempotency, abort and undo pathways, concurrency rules, failure handling, and long-term cryptographic interpretation.

These mechanisms describe how the ecosystem behaves under defined constraints, policies, and security envelopes.

1. Revocation and Trust-State Adjustment

Revocation allows automation instructions, template versions, or key material to be withdrawn after issuance.

The system uses two complementary layers.

1.1 Local Revocation (Device-Level)

Each orchestrator maintains its own revocation registry containing:

- invalidated Capsule IDs
- deprecated or blocked template hashes
- revoked signing keys
- trust-state adjustments (e.g., sender demoted or removed)

Before executing any Capsule, the orchestrator checks it against this local registry.

Local revocation is authoritative and always evaluated first.

This preserves strong receiver sovereignty.

1.2 Global Revocation via Hash Anchoring (IOTA and Solana)

For multi-organizational workflows, revocation identifiers may be optionally published as hash commitments on:

- IOTA (long-term archival integrity)
- Solana (near-real-time dissemination)

Anchors contain only irreversible hashes, e.g.:

- Capsule-ID hashes
- template-version hashes
- key fingerprints

No Capsule contents, workflow data, or Vault identifiers are ever published.

These optional anchors provide cross-organizational visibility without requiring shared infrastructure or compromising privacy.

Local policy may accept or ignore global anchors as needed.

2. Idempotency and Replay Behavior

Tasks differ in how often they should run.

Capsules may therefore specify idempotency behavior.

2.1 Idempotency Modes

- single-use
- conditional repetition (execute only if defined context changed)
- repeatable execution
- independent repetitions with separate results

2.2 Replay Evaluation

Replay logic is applied per session.

Multi-session Capsules can mix different idempotency rules within one Capsule.

3. Session Processing Model

The WRCode ecosystem distinguishes between interactive sessions and headless automation workflows.

3.1 Sequential Execution for Human-in-the-Loop

Interactive sessions typically involve:

- Agent Boxes
- Mini-Apps
- display grids
- Augmented Overlay components
- pointer guidance or UI navigation

To avoid conflicting UI states and ambiguous user focus, only one interactive session executes at a time.

Once a session completes, the environment resets before the next one begins.

3.2 Parallel Execution for Headless Automation

Non-interactive tasks may run concurrently when:

- the Capsule explicitly selects headless operation,

- no UI rendering is required,
- no user approval is pending,
- no interactive components are active.

This mode is appropriate for:

- extraction
- classification
- transformation
- autonomous multi-agent tasks

Parallelism is controlled by local resource policy.

4. Partial Execution and Execution Boundaries

Some workflows progress through multiple internal stages.

4.1 Session-Level Isolation

- Each session is isolated.

- One session may succeed while another fails.
- PoE reflects the per-session outcome.
- State changes from one session do not spill into another.

4.2 Step-Level Outcomes

Mini-Apps may define multi-step sequences, including:

- atomic updates
- partially completed steps
- non-rollbackable operations
- compensating logic (optional)

These structures help downstream systems interpret progress accurately.

5. Abortable Automations and Undo Capabilities

The architecture supports controlled interruption and rollback pathways.

5.1 Abort Handling

An automation may be aborted when:

- the user cancels an interactive workflow,
- revocation is detected during execution,
- dependencies fail,
- local or global trust-state changes occur,
- Mini-Apps or Agent Boxes signal internal abort conditions.

When an abort occurs:

1. The orchestrator halts the session at the next safe interruption point.
2. UI or overlay components close cleanly in interactive mode.
3. The partial state is recorded in PoE.
4. The environment resets before the next session begins.

Abort behavior never exceeds local policy or Vault permissions.

5.2 Undo and Reversal Behavior

Some state changes can be reverted. Reversal depends on workflow structure and the nature of the operation.

Potentially Reversible Examples

- modifications within Mini-Apps
- transient state inside Vault-A
- reversible UI interactions
- local data transformations
- internal flags or progress markers

Generally Non-Reversible

- external system calls
- irreversible cryptographic operations
- outbound transmissions
- physical-world actions

When an operation cannot be reversed, the orchestrator may:

- perform a compensating action,
- reset to a known earlier state,
- or issue diagnostic output describing the new stable state.

Undo operations may be:

- triggered manually (interactive workflows),
- triggered automatically (compensating logic),
- triggered by local policy after abort events.

5.3 PoE Representation of Abort/Undo Outcomes

PoE records:

- steps executed prior to interruption,
- aborted actions,
- reversal or compensation steps,

- final resulting state.

This provides structured interpretability for downstream parties.

6. Failure Handling, Diagnostic Output, and Dead Letter Queue

Failure handling is a core runtime component.

6.1 Dead Letter Queue (DLQ)

Capsules that cannot be processed—due to revocation, missing context, signature failure, TME expiration, or policy mismatch—are stored in an encrypted DLQ inside Vault-A or Vault-B (depending on sensitivity).

6.2 Diagnostic Capsules

If permitted, the orchestrator may issue a Diagnostic Capsule containing minimal metadata describing:

- the failure type,

- relevant policies,
- hints for remediation.

Diagnostic Capsules never expose Vault data.

7. Access Control in Multi-Session Capsules

Different sessions may specify different execution requirements.

7.1 Per-Session Policy

A session may request:

- manual approval
- partially human-in-the-loop workflows (automatic response but manual execution confirmation)
- full headless automation

7.2 Policy Resolution

If policies conflict, the most restrictive is applied.
This prevents unintended escalation.

8. Cryptographic Lifecycle and Key Rotation

Long-term interoperability requires consistent handling of cryptographic material.

8.1 Key Rotation

Key rotation refers to periodically replacing signing keys to:

- limit exposure
- respond to compromise
- comply with organizational requirements
- accommodate staff or infrastructure changes

Old Capsules remain verifiable because they reference a historical key identifier.

8.2 Long-Term Interpretation

PoE and Capsules include:

- key identifiers,
- canonical hash commitments,
- environment measurements.

This allows interpretation years later, even after keys have been rotated.

8.3 Optional Integrity Anchoring

Hashes of:

- Capsule IDs,
- template hashes,
- and key fingerprints

may be anchored on IOTA or Solana for independent timestamping and audit-level interpretability.

Anchoring does not alter execution behavior.

9. Non-Interactive Verification Pathways

For auditing or dispute resolution, a minimal verification interface may expose:

- canonical Capsule hashes
- minimal PoE metadata
- timing information
- environment measurement identifiers

Vault content remains inaccessible.

Summary

This unified architecture describes how the BEAP and WRCode system handles:

- revocation (local and blockchain-anchored),
- idempotency and replay logic,
- sequential vs. parallel session execution,
- partial and multi-step workflows,

- abort and undo behavior,
- failure routing and DLQ,
- per-session access control,
- cryptographic lifecycle including key rotation,
- non-interactive verification mechanisms.

Together, these elements form a predictable, policy-driven automation environment that supports both interactive and headless workflows while maintaining local control, privacy boundaries, and clear execution semantics.

Structured Handling of Emails, PDFs, Invoices, and Attachments in BEAP/WRCode

In BEAP/WRCode, emails, PDFs, invoices, and attachments are treated as structured data artifacts rather than executable or renderable documents. Ingestion and interaction are governed by verifiable provenance, cryptographic integrity, and orchestrator policy. The design separates semantic accessibility from document rendering and execution, minimizing attack surface while preserving automation and analysis.

Provenance, WRStamping, and Trust Establishment

WRStamped content is cryptographically bound to a WRCode account and generation context. Where hardware-backed attestation (e.g., TPM/TEE/Secure Enclave) is available, it can strengthen provenance signals and policy gating for automation decisions.

This trust establishment is used for attribution, auditability, and policy decisions. It does not change the operational rule that original emails, PDFs, and attachments are not rendered inside the protected environment.

Parsing and Canonical Capsule Representation

Independently of provenance, incoming artifacts can be transformed into a canonical Capsule representation consisting of:

- normalized text
- structural metadata (e.g., boundaries, page/section markers when available)
- cryptographic hashes and WRStamp references (if present)

Normalization uses strict extraction paths that avoid executing active document elements. The canonical representation is the basis for downstream automation, policy evaluation, optional embeddings, and user-visible inspection.

Two Ingestion Paths: Context vs. Typed Documents

The Capsule Builder supports two distinct ingestion paths, chosen by policy and/or user intent.

1) Context Mode (Unstructured / General Documents)

Context mode is intended for documents such as:

- scientific papers
- manuals and policies
- contracts (as context)
- long email threads
- mixed document collections

Behavior:

- The document is treated as context only.
- It is normalized into plain text and stored as Capsule context.
- No automatic schema-type detection is performed.

- No structured fields are created by default.
- The result is optimized for reading, reasoning, semantic search (if enabled), and reference during automation—without implying structured invoice semantics.

This mode requires no special preparation and does not attempt to infer domain-specific key-value structure.

2) Typed Extraction Mode (Invoices, Receipts, Structured Forms)

Typed extraction mode is intended for documents such as:

- invoices
- receipts
- purchase orders
- delivery notes
- standardized business forms

Behavior:

- A document type (e.g., *Invoice*) is selected or detected.
- The content is parsed into a predefined domain schema.
- Structured fields are extracted and written into Capsule fields, such as:
 - customer number / customer ID
 - invoice number
 - dates
 - totals / taxes
 - currency
 - line items (quantity, unit price, description)

Example output fields:

- `customer_number = 123`
- `invoice_id = INV-2025-00421`
- `total_gross = 1,249.00 EUR`

During extraction, the builder MAY flag missing or ambiguous fields (e.g., no customer reference) as non-binding hints to improve schema completeness. These hints do not overwrite extracted data.

Typed extraction exists specifically to enable reliable receiver-side automation (validation, booking, approval workflows) based on explicit schema fields rather than free text. Within the Capsule Builder, senders may explicitly flag content as typed (e.g., invoice, receipt, or form) to trigger schema-based extraction; unflagged content defaults to Context Mode and is embedded as normalized plain text only.

Semantic Embedding and Local Reasoning

On the receiver side, semantic embedding is optional. If enabled, the normalized text (from either Context Mode or Typed Mode) can be embedded into a local vector store using locally executed embedding models. Higher-capacity reasoning models may then be used for privacy-preserving semantic search and contextual reasoning on organization-controlled hardware.

Embedding does not alter ingestion semantics and does not affect presentation.

Presentation Model: No Rendering, No Rasterization within the protected environment:

- No original email or document is rendered
- No rasterization is performed

- No HTML/CSS is executed
- No document fonts, images, or layout instructions are processed

User-visible presentation is limited to:

- structured plain-text views
 - schema-based views (for typed documents)
 - locally defined templates for readability (logo-free, image-free)
-

External Access to Originals via WR/QR Redirection

Original emails, PDFs, and attachments may be needed for legal retention, dispute handling, or manual inspection. In BEAP/WRCode, access to originals is provided only via explicit WRCode/QR redirection:

- Originals remain inaccessible inside the protected environment.
- The UI displays a WR/QR marker for each original artifact.
- Scanning transfers a reference, not an inline render, to an out-of-network device.

- Retrieval, viewing, and archival occur outside the protected environment and are out of scope of the in-network execution boundary.

Link Handling and Network Isolation

Hyperlinks are never exposed as clickable elements inside the activated protection layer of the orchestrator. They are displayed as plaintext and links are replaced by WR/QR markers to be opened only on isolated external devices.

Capsule Routing and Response Behavior

Upon Capsule reception, the orchestrator detects the inbound channel and applies a response policy. Responses can be composed via text, speech, templates, or automations and routed back through the same channel or a policy-defined channel. Response modalities are treated uniformly as automation variants.

WRGuard and Retention

If WRGuard is enabled, original artifacts can be archived unchanged outside the operational environment to satisfy retention or evidentiary requirements. The protected environment continues to enforce text-only presentation, link isolation, and external-device access to originals.

Summary

BEAP/WRCode treats emails, PDFs, invoices, and attachments as verifiable, automation-ready data artifacts, not renderable documents. The Capsule Builder supports two distinct ingestion paths: Context Mode, which embeds fully formatted documents as plain text without schema creation, and Typed Extraction Mode, which parses invoices and business forms into explicit schema fields (e.g., customer number, totals, line items) for receiver-side automation. Original artifacts are never rendered or rasterized inside the protected environment; access is provided only via WR/QR redirection to external, out-of-network devices.

Interoperability Scope

BEAP defines interoperability at the Capsule and Context level, including the container format, mandatory fields, integrity verification, and normalized context objects such as documents and evidence payloads. Orchestration logic, execution policies, agent routing, and session semantics are intentionally out of scope. While session data may be included in a Capsule for continuity, auditing, or advisory purposes, its interpretation is implementation-specific and MUST NOT be assumed to be semantically interoperable across different systems. This separation preserves security, accountability, and system-level autonomy while ensuring reliable exchange of verifiable data.

Controlled Orchestration Interoperability (Non-Normative Design Considerations)

The following outlines a technically feasible approach for enabling *limited and controlled interoperability* of orchestration logic between selected systems. This description is provided for architectural clarity and intellectual property protection only and does not constitute a guarantee, requirement, or baseline capability of BEAP.

Canonical Orchestration Intermediate Representation (IR)

Any form of orchestration interoperability would require a canonical, provider-neutral intermediate representation (IR). Such an IR would explicitly model orchestration constructs including, but not limited to, triggers, execution steps, data flows, conditional logic, side-effects, secret handling, permission scopes, timeouts, retries, idempotency invariants, and error-handling semantics.

The IR would further enforce a strict type system and explicit capability model defining which operations a step is permitted to perform (e.g., network access, email dispatch, vault access, file system interaction). Capabilities not explicitly declared would be denied by default.

Deterministic Import and Export Pipelines

Rather than relying on semantic interpretation, interoperability would depend on deterministic transformation pipelines. For each supported orchestration system, a formally specified importer would translate a vendor-specific orchestration description into the canonical IR. Conversely, a

deterministic exporter would translate validated IR artifacts into the target system's orchestration format.

For widely adopted platforms, predefined master schemas or mapping profiles could be provided to cover common orchestration patterns. These mappings would be versioned, explicit, and limited in scope.

Assistive Use of Generative Models

Generative models may be used exclusively as assistive tooling during orchestration migration or import. Typical use cases include proposing step mappings, normalizing identifiers, suggesting heuristic correspondences, or generating an initial IR draft for review.

Under no circumstances would generative output be treated as authoritative. All IR artifacts would be subject to formal schema validation, type checking, capability enforcement, deterministic canonicalization, and—where security-relevant behavior is affected—explicit human approval. This ensures that execution traces, proof-of-execution records, and audit logs remain deterministic and reproducible.

Semantic Boundaries and Compatibility Levels

Semantic mismatches represent the primary limitation to orchestration interoperability. Differences in event models (e.g., push versus polling), delivery semantics, timing semantics, state management, and side-effect behavior cannot be reliably abstracted without explicit modeling.

Any interoperability mechanism would therefore require explicit compatibility classifications, such as *lossless*, *lossy with warnings*, or *not portable*. Transformations that cannot guarantee semantic equivalence would be restricted to advisory use or blocked entirely.

Fail-Closed Activation Model

In enterprise or high-assurance deployments, orchestration artifacts that cannot be safely or unambiguously mapped to a target system would not be automatically activated. Instead, such cases would produce a machine-readable import report detailing incompatibilities, semantic deviations, and required manual interventions. Activation would require explicit review and authorization.

Threat Classes and Mitigations

1. Malicious Document Execution (PDF / Email Exploits)

Threat:

Attackers embed malicious payloads in PDFs or emails (e.g., scripts, malformed objects, font exploits, macro-like constructs) to trigger code execution or privilege escalation during document viewing.

Mitigation:

- Emails and PDFs are not rendered as active documents by default.
 - Original PDFs are never rendered or rasterized within the protected environment and are accessible only via WR/QR redirection to external devices.
 - Emails are normalized into deterministic WR templates and displayed as structured plain text, eliminating HTML rendering, scripts, embedded objects, and external resource loading.
 - Parsing is limited to deterministic text and metadata extraction and never executes document logic, layout instructions, or embedded behaviors.
-

2. Script Injection and Active Content Abuse

Threat:

Embedded scripts, dynamic elements, or externally loaded resources execute during rendering or interaction.

Mitigation:

- All default rendering paths are non-executable by construction.
- No inline scripts, external resources, iframes, or dynamic objects are exposed in the orchestrator interface.

- Semantic representations used for automation are strictly separated from rendering logic and cannot trigger execution.
-

3. Phishing and Malicious Link Interaction

Threat:

Users are tricked or induced into interacting with malicious links embedded in emails or documents, leading to credential theft, malware delivery, or data exfiltration.

Mitigation:

- Hyperlinks are never exposed as clickable elements within the active WRGuard protection layer.
- All links are reduced to plaintext references and replaced with QR codes or WR Codes.
- Link interaction is explicitly displaced to external devices outside the company network, preventing untrusted network interaction within the operational environment.

4. External Link-Viewer Device Risks

Threat:

Malicious links exploit the device used for external link inspection or attempt lateral movement into trusted systems.

Mitigation:

- Link viewing occurs exclusively on external devices outside the company network.
- These devices operate under strong sandboxing constraints, including restricted process isolation, limited permissions, and the absence of trusted communication channels back to the orchestrator.
- Optional AI-based link analysis (e.g., phishing detection, structural URL analysis, reputation checks) may be performed on the external device as an advisory signal only.
- External link-review devices are treated as low-trust endpoints and hold no orchestrator credentials, secrets, or execution privileges.

5. Capsule Tampering, Replay, and Integrity Attacks

Threat:

Attackers modify Capsule contents, inject unauthorized fields, replay previously valid Capsules, or attempt to bypass execution constraints.

Mitigation:

- Every Capsule is subject to mandatory cryptographic verification prior to any processing.
 - Verification includes:
 - hash validation,
 - structural integrity checks,
 - schema compliance validation,
 - WRStamp verification,
 - policy and capability enforcement.
 - Capsules that fail verification are unconditionally rejected.
 - There is no downgrade path, fallback mode, partial execution, or exception handling.
-

6. Unauthorized Automation Execution or Privilege Escalation

Threat:

A Capsule attempts to trigger automation steps, data access, or side effects beyond its declared scope.

Mitigation:

- Capsule execution is governed by explicit, pre-declared capabilities and policies defined at build time and enforced at runtime.
 - Capsules cannot escalate privileges or access undeclared resources.
 - Automation operates exclusively on verified Capsule representations, not on raw or executable originals.
-

7. Archive Storage Separation and Retention Risks

Threat:

Archived originals become a secondary attack surface or are coupled to runtime execution or link-interaction paths.

Mitigation:

- Archiving is never performed on the company or orchestrator network.
- By default, archival may be performed on the same external device used for link viewing, provided that:
 - archiving is sandboxed,
 - no execution of archived content is permitted,
 - no trust or execution relationship exists with the orchestrator runtime.
- For high-risk or high-assurance environments, it is recommended to fully separate:
 - the external link-viewer device, and
 - the archival storage system
onto distinct, hardware-isolated external devices.
- Archives may be encrypted at rest.

Archiving is not required at runtime. Instead:

- Archival can be performed asynchronously or at configurable time intervals.
- Runtime archiving occurs only when explicitly required by policy or regulatory constraints.

8. Cross-Channel Injection and UI Confusion Attacks

Threat:

Attackers exploit differences between email, PDF, and messaging channels to create misleading UI states or impersonate trusted senders.

Mitigation:

- All inbound content is normalized into Capsules with explicit provenance metadata.
- Rendering behavior is derived from Capsule verification, WRStamp validation, and attestation state, not from channel appearance or sender claims.
- Deterministic templates and plain text views remove channel-specific UI affordances that could be abused.

9. Data Exfiltration via Automation or Embedding

Threat:

Sensitive content is unintentionally leaked through automation pipelines, semantic embeddings, or reasoning components.

Mitigation:

- Semantic embedding is explicitly opt-in and under receiver control.
 - All embeddings and reasoning operations are executed on organization-controlled hardware.
 - Automation operates exclusively on normalized, verified Capsule representations, not on executable originals.
-

10. Parser Abuse and Resource Exhaustion

Threat:

Specially crafted documents trigger parser failures or excessive resource consumption.

Mitigation:

- Parsing is limited to deterministic text and metadata extraction.
 - Large-context ingestion is processed incrementally with bounded resource usage.
 - Original files are not required for runtime automation and may remain offline or archived.
-

11. Trust Spoofing and Content Impersonation

Threat:

Attackers attempt to impersonate trusted senders or inject content that appears legitimate.

Mitigation:

- WRStamping cryptographically binds content to verified WR Code accounts.
 - Hardware-backed attestation links content generation to an untampered execution environment.
 - Trust decisions are derived from verifiable cryptographic signals, not visual appearance or sender identity alone.
-

Residual Risk and Design Trade-offs

BEAP/WRCode prioritizes attack surface reduction, strict verification, and isolation over convenience.

As a result:

- Original rendering is restricted and policy-controlled.
- Network interaction is displaced to external, isolated devices.
- Archival storage is decoupled from runtime execution and link inspection.

- Enterprise deployments may explicitly block unsupported email providers to prevent incomplete protection states.
- Capsule verification enforces a fail-closed execution model with no exceptions.

Residual risks are primarily procedural (e.g., user behavior on external devices) rather than architectural.

Summary

By enforcing mandatory Capsule verification, provider-agnostic Capsule construction, secure views for emails and attachments (within supported email providers), isolated external link interaction, configurable external archiving, and enterprise-level provider controls, BEAP/WRCode constrains document-, link-, and automation-based attack vectors without relying on heuristic detection or real-time content execution. Only fully verified Capsules participate in automation; all other content is rejected outright.

Deterministic Structured Document Processing for BEAP-Compliant Capsules

1. Introduction

In BEAP-based automation systems, documents such as PDFs are treated as structured data carriers rather than passive attachments. The objective is to enable predictable, auditable, and software-agnostic automation while allowing senders to participate without shared templates, proprietary tooling, or knowledge of the receiver's internal systems.

To achieve this, BEAP defines a minimal document compliance rule that allows heterogeneous document sources to be converted into deterministic, machine-actionable data structures embedded directly into capsules.

2. BEAP Structured Document Compliance Rule

A document is considered BEAP-compliant if all relevant information is expressed as explicit key-value pairs, for example:

Customer Number: 7483487

Invoice Date: 2025-01-15

Total Amount: 1,249.00 EUR

This rule applies regardless of document origin, including digitally generated PDFs, scanned documents processed via OCR, or other structured formats rendered as PDFs. Semantic

interpretation relies exclusively on labeled fields rather than layout, formatting, or positional heuristics.

By adhering to this rule, a sender only needs to understand the BEAP standard to create documents that can be reliably processed by automation systems downstream.

3. Deterministic Parsing and Capsule Builder Pipeline

The BEAP capsule builder operates in a strictly deterministic manner. No large language models, probabilistic inference, or generative reasoning are involved during capsule construction.

The processing pipeline consists of:

1. Text Acquisition

Document content is obtained via direct text extraction or OCR with layout normalization.

2. Field Detection

Explicit key–value patterns are identified using deterministic parsing rules.

3. Canonical Normalization

Field labels are mapped to canonical identifiers using predefined vocabularies and alias mappings to ensure semantic consistency.

4. Type Assignment

Values are normalized into explicit data types (e.g., integer, date, currency, string) using deterministic and locale-aware rules.

The output is a reproducible, auditable representation of the document's structured content.

4. Role of Machine Learning and Reinforcement Learning

Machine Learning (ML) and Reinforcement Learning (RL) are used exclusively in the case that user consent is mandatory for the deterministic results. Their role is limited to:

- identifying recurring domain-specific field patterns,
- refining alias mappings and normalization rules,
- improving robustness across document variants.

The outcome of this process is a static, deterministic parser configuration. Once deployed, the capsule builder does not rely on ML, RL, or LLM-based reasoning, ensuring predictable execution behavior.

5. Error Flexibility and Manual Overrides

BEAP document processing is intentionally error-tolerant:

- Missing or unresolved fields do not invalidate a capsule.
- Partial schemas remain valid as long as they fall within the declared scope.
- Fields may be augmented later or manually overridden by authorized users or systems.

Manual correction is considered a first-class capability. Deterministic automation must not prevent human intervention or contextual adjustments. This approach avoids brittle workflows and reflects real-world document variability.

6. Deterministic Type-Schema Generation and Capsule Embedding

Extracted key-value pairs are transformed into a custom, deterministic type schema that represents the semantic structure of the document. This schema is:

- data-only and non-executable,
- explicitly typed,
- extensible for domain-specific fields,

- versionable and auditable.

The type schema is embedded into the capsule during the capsule builder process and serves as the canonical interface between document input and automation logic.

7. Adapter-Based Automation Integration

Typed schemas embedded in capsules can be connected to custom adapters that interface with specific software products, APIs, or internal systems.

This architecture decouples document creation from software integration:

- Senders only need to follow the BEAP document compliance rule.
- Receivers map schema fields to their software landscape through adapters.

As a result, BEAP-compliant documents can trigger or support custom automation across heterogeneous environments without tight coupling or pre-shared formats.

8. Optional WR Code Error Reporting for Continuous Improvement

To support continuous improvement of parser quality and domain coverage, BEAP implementations may optionally generate and share structured error reports via WR Code mechanisms.

Participation in error reporting is strictly opt-in. Users can enable this feature to adapt their systems to specific domain requirements and to contribute anonymized feedback that improves parser definitions for all users over time.

When enabled, error reports may include:

- unresolved or ambiguous field labels,
- normalization or type assignment conflicts,
- schema gaps for previously unseen domain fields,
- parser version and configuration identifiers.

Error reports are privacy-preserving, non-executable, and detached from runtime automation. Shared data is used exclusively for offline refinement of deterministic parsers and vocabularies and does not influence capsule construction at execution time.

9. Determinism, Auditability, and Safety

By enforcing a strict separation between learning (offline) and execution (runtime), the system ensures:

- deterministic capsule construction,
- predictable automation behavior,
- auditable data transformations,
- and clear safety boundaries between data and execution logic.

Documents remain data; automation logic remains explicit; human control remains intact.

10. Conclusion

By combining a minimal structural document rule, deterministic capsule construction, error-flexible schema handling, adapter-based integration, and opt-in WR Code error reporting, BEAP enables scalable and domain-adaptive document automation without sacrificing predictability, interoperability, or human oversight.

High-Assurance Orchestrator Mode

A WRCode-Governed, Hardware-Attested, Temporally Enforced Execution Architecture

High-Assurance Mode is an operational configuration of the WRCode orchestrator designed for high-risk environments such as critical infrastructure, defense, financial institutions, and regulated governmental systems. In this mode, the orchestrator becomes a policy-bound execution appliance in which all permissible behaviors are defined exclusively by externally sealed WRCode Access Control Policies (ACP). These policies are cryptographically anchored and validated using hardware attestation. Handshakes in this mode require full dns verification on both sides.

Any deviation—including attempts to disable controls, modify execution paths, or alter device state—is detectable, log-visible, and forensically auditable via the attestation pipeline, Proof-of-Execution (PoE) logs, and Time-Managed Execution (TME/TPE) subsystem.

1. Hardware Attestation Foundations

WRCode-Enforced Attestation Requirement

Activation of High-Assurance Mode requires verified hardware attestation. WRCode enforces that the orchestrator is booted and executed within an attested, trustworthy environment. WRCode does not

prescribe specific vendors; instead, it validates attestation signals and activates High-Assurance Mode only when the device state matches the expected trust configuration.

Supported widely known attestation mechanisms include:

- TPM 2.0 – measured boot and platform integrity evidence
- Intel SGX – enclave-based isolation and attestation
- ARM TrustZone – hardware-isolated secure execution world
- Apple Secure Enclave – isolated cryptographic processor
- AMD SEV/SNP – attested memory integrity in virtualized systems

If attestation fails or deviates from expected measurements, High-Assurance Mode remains inactive and the deviation is recorded in the PoE ledger.

2. Time-Managed Execution (TME/TPE)

Temporal Enforcement as a Security Primitive

TME/TPE integrates enclave-controlled temporal guarantees into the system's trust boundary. A trusted hardware clock validates all time-bound policy transitions, unlock capsules, and privileged workflows.

Key Components

Trusted Enclave Clock

Immune to host OS manipulation, used to verify temporal assertions.

Temporal Validity Checks

Unlock tokens and policy deltas must match their declared validity intervals.

Expired or replayed items are rejected and logged.

Deterministic Expiration

When a privileged window ends, the baseline policy is restored automatically.

Enforcement cannot silently persist beyond the declared time.

Any clock tampering or timing inconsistency appears as a measurable deviation in TME/TPE logs.

3. Outbound Network Restriction Under WRCode Policy

High-Assurance Mode enforces strict outbound connectivity rules:

- Only destinations defined in the ACP are reachable.
- All unauthorized outbound attempts are blocked.
- Every network event—successful or rejected—is recorded in the PoE ledger.

If enforcement were altered or bypassed, discrepancies appear as:

- unexpected outbound attempts,
 - missing entries in the PoE log,
 - or attestation measurements inconsistent with expected policy.
-

4. Comprehensive Network Logging (Policy-Dependent)

When enabled, the orchestrator logs network interactions in a structured, immutable PoE ledger:

- Only normalized metadata is recorded (no payload content).
 - Integrity can optionally be anchored to external systems (e.g., Solana, IOTA).
 - Attempts to remove, alter, or suppress logs create detectable anomalies.
-

5. USB, Peripheral, and Device-Class Governance

Complete Allocation or Complete Disablement

High-Assurance Mode extends peripheral governance to all device classes, not only USB ports. This includes:

- USB ports and hubs
- CD-ROM / DVD drives
- External storage interfaces (SATA/eSATA)
- Thunderbolt and Lightning adapters
- Docking stations
- Legacy ports such as RS-232/serial, parallel, FireWire
- Removable wireless modules (Wi-Fi adapters, LTE dongles, Bluetooth sticks)
- SD-Card readers and similar removable media interfaces

Enforcement Logic

- Any device interface not explicitly allocated in the ACP is automatically disabled.

- Allowed devices must match known identity or configuration hashes.
- Inactivity timers may enforce auto-disablement.
- Attempted use of unallocated or disabled interfaces creates PoE log entries and potential attestation mismatches.

Changes to device availability (e.g., enabling an unused Lightning port, activating a CD-ROM drive, or connecting an unapproved docking station) become immediately visible as:

- unexpected peripheral enumeration events,
- missing or malformed PoE sequences,
- attestation-state drift,
- or unauthorized device activation attempts.

The model ensures complete allocation or complete disablement, with no silent activation pathways.

6. Mandatory Secure Rendering Overlay

In High-Assurance Mode, WRGuard's secure rendering view is mandatory unless specifically permitted by the ACP.

- All PDFs, emails, and structured attachments are displayed in a standardized secure view.
- Binary originals remain inaccessible.
- Attempts to bypass the secure overlay are blocked and logged.

Tampering with rendering controls manifests as:

- unapproved binary file access attempts,
 - deviations between expected and observed rendering behavior,
 - attestation inconsistencies.
-

7. Controlled Unlock Pathways

WRScan + Special WRCode Roles + Attested Hardware

High-Assurance Mode permits controlled unlock operations for specific WRCode-defined roles (e.g., auditors, compliance officers, supervisors). These unlocks are:

- time-limited
- attestation-validated

- externally sealed
- fully PoE-recorded

Unlocking requires:

7.1 Verified Hardware Attestation

The enclave verifies that the device state matches the expected attestation baseline.

7.2 WRScan-Based Unlock Capsule

- A WRStamped unlock capsule is scanned using Beap Scan.
- Contains the role identity, policy deltas, and a declared TME validity window.
- The enclave validates authenticity and checks for expiration or replay.

7.3 Temporal Enforcement (TME/TPE)

- Privileged state remains active only for the approved time window.
- The system reverts to its baseline automatically when the window closes.
- All unlock operations are PoE-logged.

Any attempt to misuse unlock functionality—expired tokens, incorrect roles, altered device state—produces visible deviations in the PoE and attestation logs.

8. Enforcement Visibility and Forensic Assurance

All Tampering Produces a Trace

High-Assurance Mode is designed so that any change, deviation, or tampering attempt generates observable artifacts in the combined attestation, PoE, and TME/TPE logs.

Examples include:

- unauthorized activation of unused device classes (USB, CD-ROM, Lightning adapters, docking stations, etc.)
- unexpected outbound network attempts
- missing log entries
- inconsistent attestation measurements
- manipulation of rendering protections
- replay or misuse of unlock capsules

- time inconsistencies detected by TME/TPE

The model prioritizes detectability, verifiability, and forensic accountability, rather than relying on absolutist assumptions.

Summary

High-Assurance Mode provides a hardware-attested, ACP-governed, temporally enforced execution environment suitable for high-risk organizations. Key characteristics include:

- Mandatory, WRCode-enforced hardware attestation (TPM, SGX, TrustZone, Secure Enclave, AMD SEV/SNP)
- Strict ACP-governed behavior for network, peripherals, and document rendering
- Complete allocation-or-disablement policy for all device classes
- Non-overrideable secure rendering overlay
- Time-limited unlock sequences controlled via WRScan and attested roles
- Comprehensive PoE logging, surfacing all deviations
- Clear forensic visibility whenever enforcement is altered, bypassed, or tampered with

This model supports environments that require auditable security, deterministic behavior, and verifiable system integrity under all operational conditions.

Automatic Capsule Builder: Reasoning-Guided, Deterministic, and Encrypted Capsule Construction in the WRCode Orchestrator

1. Introduction

WRCode Capsules are verifiable, schema-bound automation units designed for secure machine-to-machine and cross-organizational workflows. They may be created manually by the user or automatically by the orchestrator. Both pathways rely on a single deterministic Capsule Builder engine, ensuring uniform schema validation, cryptographic protection, and structural consistency.

Although the builder itself performs no reasoning, its behaviour is guided by the orchestrator's reasoning output. Reasoning may propose role requirements, approval policies, risk levels, preferred templates, or delivery options. These values are then injected into the deterministic builder through controlled mappings and overwrite rules.

This article presents the unified builder architecture, the automatic builder's configuration, session integration and overwrite mechanism, encryption pipeline, and the handling of built-in system capsules such as Proof-of-Execution (PoE).

2. Unified Capsule Builder Architecture

2.1 One Engine for All Capsule Types

A central property of WRCode's design is that:

- Manual Capsule Builder
- Automatic Capsule Builder
- System Agents (PoE, diagnostics, state reports)

all invoke the very same deterministic builder engine.

This ensures that Capsules generated interactively or automatically — including system-level capsules — share:

- identical template semantics
- identical canonicalization
- identical encryption handling
- identical signature blocks
- identical delivery mechanisms

2.2 Invocation Paths

| Capsule Type | Initiator | Uses Builder Engine | User Setup Required |
|-----------------------------------|-------------------|---------------------|---------------------|
| Manual Capsule | User | Yes | Yes |
| Automatic Capsule | Execution Section | Yes | Yes |
| System Capsule (PoE, diagnostics) | Orchestrator | Yes | No (preconfigured) |

The builder's behaviour is shaped by structured inputs (reasoning output, workflow results, trigger metadata), but never by uncontrolled code execution.

3. Position in the Orchestrator Pipeline

The orchestrator produces structured outputs after evaluating triggers, running local reasoning, and applying workflows:

Trigger → Listener

→ Agent reasoning

→ Workflows

- Execution result (structured)
- Report Targets (Agent Box, Capsule Builder, System Agents)

If the Capsule Builder is selected as a report target, the execution result is converted into a Capsule using deterministic mapping rules and optional reasoning-guided decisions.

4. Reasoning-Guided but Deterministic Construction

The Capsule Builder does not call an LLM and does not contain logic that can make autonomous decisions. However, the orchestrator's reasoning layer may compute:

- required roles and permissions
- risk classifications
- delivery policy (draft, auto-send, approval required)
- which Capsule Template to use
- escalation rules
- contextual indicators (urgency, SLA state)

These structured results become inputs to the builder via:

- field mappings
- template selection rules
- session overwrite rules
- delivery configuration

Thus the system combines dynamic reasoning with deterministic, auditable construction.

5. Capsule Templates

A Capsule Template defines:

- fields and field types
- schema constraints
- default values
- encryption policies
- signing requirements

- allowed session overwrite locations
- optional dynamic fields guided by reasoning

Templates are versioned and can be reused, cloned, extended, or locked depending on organizational needs.

6. Automatic Capsule Builder Configuration

When the Automatic Capsule Builder is selected as a report target inside an Execution Section, the user configures:

6.1 Build Timing

`on_finish`

`on_error`

`on_condition`

`always`

Conditional rules operate on structured agent/workflow output.

6.2 Field Mapping

The user defines how template fields are populated:

- from agent reasoning output
- from workflow results
- from trigger context
- from literals
- from deterministic functions (e.g., date arithmetic, counters)

Example:

```
{  
  "field": "roles",  
  "source": { "type": "agent_output", "path": "policy.proposed_roles" }  
}
```

6.3 Template Selection

Reasoning may guide template selection:

if risk = high → use approval_chain_v2

else → use approval_simple_v1

The builder then enforces schema compliance.

6.4 Delivery and Send Mechanism

Capsules may be:

- auto-sent
- stored as drafts
- marked as requiring manual approval

6.4.1 "Reply-To" Condition and Wizard-Assisted Setup

If the user enables reply-to behaviour, the builder automatically tags the Capsule with:

- the channel used for the incoming request
- the identifiers needed for routing back to that channel

The system then triggers a wizard that guides the user through the technical steps required to enable actual sending for that channel (e.g., email OAuth, BEAP endpoint pairing, messenger API credentials).

If sending is not configured, the Capsule remains valid but is stored locally until transmission becomes possible.

6.5 Capsule ID Strategy

Options include:

- UUID
- timestamp-based ID
- snowflake-style monotonic IDs
- user-defined prefix rules

7. Encryption Model

All Capsules undergo encryption before signing.

This ensures privacy and tamper-evident structure even inside distributed workflows.

7.1 Encryption Pipeline

Before signing:

1. Capsule payload is serialized and normalized.
2. Session object is attached.
3. Overwrites are applied deterministically.
4. Payload is encrypted with a local encryption key (per-device or per-workflow key).
5. Encrypted payload is embedded in the Capsule.

7.2 Decryption Rules

Capsules are decrypted only if:

- the receiver's device holds the appropriate decryption key,
- or a mutually established handshake allows temporary key exchange,
- or the Capsule is a system capsule (e.g., PoE), which may use a different encryption policy (e.g., integrity-only Capsule with encrypted metadata block).

Encryption is independent of template, reasoning, or sender identity.
It is a core property of the Capsule format.

8. System Agents and Built-In Automatic Capsule Generation

System Agents automatically invoke the builder engine with predefined templates and policies.

8.1 Proof-of-Execution (PoE) Capsules

PoE capsules provide:

- execution hash trees
- performance metadata
- timestamps
- session fingerprints
- workflow lineage

They require no setup and cannot be disabled without affecting system integrity.

8.2 Diagnostic and Operational Capsules

These include:

- error reports
- state transitions
- channel-binding updates
- policy enforcement events

They use minimal locked templates to ensure consistency across deployments.

9. Session Attachment

Every Capsule includes a Session Object capturing:

- session start and end timestamps
- execution mode
- contextual metadata
- workflow selections
- policy state

- applied reasoning parameters

This ensures verifiable provenance.

10. Controlled Real-Time Session Overwrites

Certain session fields may be overwritten using structured reasoning output.

Example:

```
{  
  "target": "session.context.urgency",  
  "source": "agent_output.inference.urgency"  
}
```

Properties:

- only whitelisted fields may change
- overwrite sources must be structured JSON fields
- overwrites occur before encryption and signing
- the final Capsule is deterministic given identical inputs

This enables dynamic high-level behaviour (e.g., urgency, multi-role approvals) without compromising reproducibility.

11. Internal Build Pipeline

The unified builder performs the following deterministic steps:

1. Load template
2. Construct capsule skeleton
3. Apply field mappings
4. Attach session object
5. Apply reasoning-guided session overwrites
6. Validate against template schema
7. Canonicalize
8. Encrypt payload
9. Compute hash
10. Cryptographically sign Capsule

11. Optionally anchor hash to Solana/IOTA
12. Deliver Capsule (if sending is enabled) or store locally

System Agents run through the exact same pipeline with predefined templates.

12. Security Considerations

- Builder performs no LLM reasoning
- No dynamic code execution is permitted
- Encryption is mandatory for payload protection
- Delivery is restricted unless properly configured (wizard-guided for reply-to channels)
- All session overwrites are schema-restricted
- Capsules remain reproducible and legally defensible

13. Conclusion

The Automatic Capsule Builder extends the WRCode platform by enabling reasoning-guided, encrypted, deterministic Capsule construction as part of orchestrated workflows. Reasoning may dynamically propose roles, permissions, templates, and behavioural flags, while the builder enforces structural correctness, controlled overwrites, and cryptographic integrity. System Agents such as PoE

capsules demonstrate that capsule construction is not merely a user-level tool but a fundamental part of WRCode's operational and auditability model.

The unified approach ensures that every Capsule — whether manually authored, generated automatically, or produced by internal system logic — adheres to the same secure, verifiable, and reproducible construction pipeline.

Adapter Architecture and Orchestrator Integration for BEAP-Based Enterprise Automation

1. Purpose and Design Goals

Adapters in BEAP-based systems provide a deterministic, auditable integration layer between capsules containing structured data and external software systems such as ERP, accounting, logistics, or compliance platforms.

The adapter architecture is designed to satisfy enterprise and regulatory requirements, including:

- clear separation of responsibilities,
- least-privilege data handling,
- deterministic execution behavior,
- auditability and replayability,
- controlled extensibility,

- and verifiable software integrity.

Adapters are not decision engines or trust authorities. They execute deterministic transformations under orchestrator control.

2. Architectural Placement within the Orchestrator

Adapters are implemented as a dedicated integration subsystem within the orchestrator and are divided into two strictly separated planes.

2.1 Control Plane: Adapter Registry and Execution Planning

The control plane resides within the trusted core of the orchestrator and is responsible for:

- adapter discovery and capability matching,
- policy evaluation and approval enforcement,
- adapter version pinning and compatibility checks,
- execution plan construction (preview, validate, commit),
- fallback selection and documentation.

All decisions regarding *whether, how, and under which conditions* an adapter may be executed are made exclusively within this plane.

2.2 Data Plane: Adapter Execution Host

Actual adapter execution occurs within an isolated adapter execution host, implemented as a separate process, sandbox, or container.

The execution host is responsible solely for:

- receiving a fully provisioned execution context,
- performing deterministic data transformation or scoped external interaction,
- returning execution results and receipts.

The adapter execution host has no autonomous access to the orchestrator core, the vault, or unrestricted network resources. This isolation forms a mandatory enterprise security boundary.

3. Adapter Definition

An adapter is defined as a deterministic, versioned integration module that:

- consumes a strictly defined input contract,
- produces a predictable output or scoped system interaction,
- does not infer intent or perform autonomous reasoning,

- does not independently select or retrieve data sources,
- and does not escalate privileges.

Adapters operate exclusively on the execution context explicitly provided by the orchestrator.

4. Adapter Capability Declaration

Each adapter must expose a formal capability declaration, including:

- supported input type schemas,
- supported output formats or target systems,
- required and optional input fields,
- supported operations (e.g., preview, validate, commit, rollback if applicable),
- adapter version and compatibility constraints.

This declaration enables deterministic adapter selection and prevents implicit or heuristic integration behavior.

5. Capsule-Side Format Intent Declaration

Capsules may optionally declare format intent, such as:

- preferred output formats,
- acceptable alternative formats,
- target system identifiers,
- fallback allowance indicators.

These declarations are descriptive only. They express intent, not authority. Final execution decisions are made by the orchestrator.

6. Adapter Selection and Fallback Handling

Adapter selection is performed exclusively by the orchestrator control plane based on:

- capsule-declared intent,
- adapter capability declarations,
- enterprise policies,
- and system availability.

If no preferred adapter is available, a compatible fallback adapter may be selected if explicitly permitted. All fallback executions are explicitly marked and recorded as degraded or alternative execution paths.

Fallback is a defined execution state, not an implicit failure condition.

7. Vault Data Usage and Provisioning Model

Enterprise automation frequently requires the inclusion of internal or confidential data stored in a secure vault (e.g., master data, accounting references, credentials).

To support this while preserving regulatory control, BEAP enforces a provisioning-based vault access model:

- Adapters do not autonomously retrieve data from the vault.
- Vault access is performed exclusively by the orchestrator.
- Retrieved data is provisioned into the adapter execution context according to explicit policies, roles, and approvals.

This distinction ensures that:

- data scope is explicitly defined,
- data minimization can be enforced,
- and all access is auditable.

Vault-provisioned inputs may include:

- raw values (only when strictly required),
- derived or transformed values,
- scoped tokens or credentials,
- cryptographic proof or match results.

Adapters consume vault-derived data only as part of the execution context and cannot request or expand access independently.

8. Execution Context and Input Binding

Before adapter invocation, the orchestrator constructs a bounded execution context that may include:

- capsule-derived structured data,
- explicitly approved vault bindings,
- adapter-specific input mappings,
- execution metadata (timestamps, adapter version, scope identifiers).

Adapters operate strictly within this context and have no visibility beyond it.

If required inputs are missing, adapters may signal a data requirement, prompting the orchestrator to evaluate whether additional data may be provisioned or must be requested through external or human interaction.

9. Approval Gates and State Transitions

Authorization and approval are modeled as explicit state transitions, not UI-triggered events.

A typical execution flow:

1. Adapter produces a preview or proposed target representation.
2. Orchestrator evaluates approval requirements.
3. Required approvals are verified.
4. Orchestrator authorizes commit execution.
5. Adapter performs the scoped commit operation.

Adapters are unaware of approval logic and do not manage authorization state.

10. Determinism, Replayability, and Auditability

Adapter execution is deterministic:

- identical inputs and adapter versions yield identical outputs,
- executions are replayable for verification and audit,
- adapter versions are explicitly pinned and recorded.

All executions are recorded in Proof-of-Execution (PoE) logs, including adapter identity, version, execution scope, approval references, fallback indicators, and outcomes.

11. Hardware Attestation and Software Integrity

In enterprise-grade deployments, hardware-based attestation mechanisms may be used to verify that:

- the orchestrator,
- the adapter execution host,
- and the adapter binaries or bundles

are running verifiably original, untampered software.

Attestation binds execution to known measurements or trusted configurations and makes deviations visible in execution metadata and PoE logs. Attestation increases assurance but does not grant additional privileges.

12. Security and Compliance Constraints

Adapters must adhere to strict constraints:

- no autonomous data acquisition,
- no privilege escalation,
- no hidden side effects,
- no execution outside orchestrator authorization.

Adapters are treated as replaceable, certifiable components, enabling controlled upgrades, third-party validation, and regulatory review without altering the trust model.

13. Conclusion

By separating vault access from adapter execution, enforcing deterministic behavior, isolating execution environments, and optionally binding execution to hardware attestation, BEAP enables secure, auditable, and enterprise-ready integration with external systems. This architecture supports advanced automation use cases while preserving compliance, transparency, and human oversight.

1. Normative Data-Request Pattern Specification

1.1 Purpose

The Data-Request Pattern defines a controlled mechanism by which an adapter may signal missing or insufficient input data without gaining autonomous access to protected data sources, including vaults.

The pattern preserves determinism, auditability, and least-privilege properties while enabling dynamic, enterprise-grade workflows.

1.2 Core Principle

Adapters may declare data requirements, but they may not select, retrieve, or expand data access autonomously.

All data acquisition decisions remain with the orchestrator.

1.3 Data-Request Declaration

An adapter MAY emit a Data-Request Declaration when required input data is unavailable or incomplete.

A Data-Request Declaration MUST include:

- required field identifiers (canonical schema IDs),
- purpose of use (e.g., accounting booking, tax classification),

- required data type and scope,
- execution phase (preview / validate / commit),
- adapter identity and version.

The declaration **MUST NOT**:

- reference storage locations,
- request unrestricted datasets,
- include executable logic.

1.4 Orchestrator Evaluation

Upon receiving a Data-Request Declaration, the orchestrator **MUST**:

1. evaluate applicable policies and trust level,
2. verify role- and approval requirements,
3. determine whether the requested data may be:
 - provisioned from the vault,
 - derived from existing context,

- requested via human interaction,
- or denied.

All outcomes MUST be explicitly logged.

1.5 Execution Restart Semantics

If additional data is provisioned, the orchestrator MAY restart the adapter execution with a new execution context.

Execution contexts are immutable once issued.

This ensures:

- deterministic replay,
- explicit state transitions,
- and clean audit trails.

2. Reference Flow: Invoice Capsule → Accounting → Approval → Commit

This section provides a normative end-to-end reference flow illustrating adapter execution, vault provisioning, approval gating, and commit.

2.1 Input: Invoice Capsule

An incoming capsule contains:

- structured invoice schema (from PDF parsing),
- supplier reference,
- monetary values,
- optional format intent:
 - preferred: ACCOUNTING_POSTING
 - fallback allowed: true.

The capsule contains no internal accounting data.

2.2 Adapter Selection (Preview Phase)

The orchestrator:

1. normalizes the schema,
2. matches declared intent against adapter capabilities,
3. selects an accounting adapter (version-pinned),
4. executes the adapter in preview mode.

The adapter generates:

- proposed booking entries,
- but marks missing fields (e.g., cost center, tax key).

2.3 Data-Request Trigger

The adapter emits a Data-Request Declaration for:

- cost center,
- tax classification.

The adapter does not specify how or where these values are stored.

2.4 Vault Provisioning

The orchestrator:

1. evaluates policy (invoice booking allowed),
2. verifies role (e.g., accounting system role),
3. retrieves only required fields from the vault,
4. provisions them into a new execution context.

Vault access and field scope are logged.

2.5 Approval Gate

Before commit:

- the orchestrator evaluates approval requirements,
- e.g.:
 - employee → supervisor → CFO,
- approvals are collected and recorded as state transitions.

No adapter execution occurs during approval.

2.6 Commit Execution

After approval:

1. the orchestrator authorizes commit,
2. the adapter executes a deterministic posting operation,
3. the external system confirms the transaction.

The adapter returns a receipt (e.g., posting ID).

2.7 Proof-of-Execution Sealing

The orchestrator records:

- capsule hash,
- adapter identity and version,
- vault fields used (by ID, not value),
- approval references,
- execution outcome.

This record forms the immutable Proof-of-Execution.

3. Threat Model Box: Vault–Adapter Boundary

3.1 Threat Scope

This threat model addresses risks related to:

- unauthorized data access,
- privilege escalation,
- opaque data usage,

- and non-deterministic execution paths.

3.2 Explicitly Mitigated Threats

T1: Adapter over-reads sensitive data

→ Mitigated by orchestrator-only vault access and explicit provisioning.

T2: Adapter escalates privileges

→ Mitigated by fixed execution context and absence of authorization logic.

T3: Hidden data exfiltration

→ Mitigated by network egress control and execution host isolation.

T4: Non-reproducible execution

→ Mitigated by immutable execution contexts and version pinning.

3.3 Trust Boundary Definition

- The vault is accessible only within the orchestrator trust boundary.
- Adapters operate outside this boundary.
- Data crosses the boundary only through explicit, logged provisioning steps.

3.4 Hardware Attestation Impact

When hardware attestation is enabled:

- the orchestrator verifies runtime integrity,
- adapter binaries are verified as original software,
- trust state is recorded in execution metadata.

If attestation fails or is unavailable:

- execution MAY continue based on policy,
- but reduced trust state is explicitly logged.

3.5 Residual Risk Disclosure

The system does not claim absolute prevention of misuse.

Instead, all deviations, policy relaxations, and trust degradations are explicitly observable in execution logs.

This aligns with regulatory expectations for transparency rather than claims of impossibility.

Architectural Invariants and Design Intent for BEAP-Based Capsule Automation

1. Scope and Intent

This article describes the architectural intent, boundaries, and design invariants of a BEAP-based capsule, orchestrator, and adapter system.

It does not claim completeness, correctness, or universal applicability. Instead, it documents the explicit design choices that shape system behavior and limit ambiguity for implementers, integrators, and reviewers.

The goal is to provide a clear technical foundation that can evolve over time without relying on implicit assumptions.

2. Explicit Non-Goals

The system is intentionally not designed to:

- perform autonomous business decision-making,
- infer semantics beyond explicitly declared schemas,
- modify its own execution logic at runtime,
- operate as a goal-seeking or self-directing agent,

- replace human responsibility or organizational accountability,
- or assert correctness of external systems or data sources.

Automation is limited to explicitly configured, deterministic execution paths as defined by the orchestrator and adapter contracts.

3. Failure and Non-Completion as Expected States

The architecture explicitly allows for non-completion and partial execution.

A capsule execution may:

- terminate without commit,
- remain unresolved due to missing data or approvals,
- be aborted by policy,
- or pause indefinitely awaiting external input.

These outcomes are treated as normal system states, not exceptional conditions. No assumption is made that automation must always complete.

4. Time as an Explicit Execution Parameter

Time is treated as an explicit parameter, not an implicit guarantee.

Execution behavior may be influenced by:

- execution windows,
- approval validity periods,
- vault binding lifetimes,
- adapter version lifetimes.

Expired or stale elements are handled through policy and visibility rather than automatic renewal or hidden retries.

5. Schema Authority and Evolution Intent

Schemas are treated as explicit, versioned artifacts.

Design intent includes:

- explicit schema authorship,
- immutable schema versions once referenced,
- adapter bindings to concrete schema versions,

- forward evolution without retroactive reinterpretation.

This approach aims to reduce ambiguity and unintended semantic drift but does not claim to eliminate it in all implementations.

6. Adapter Behavior Visibility

Adapters are expected to operate within declared capabilities and input contracts.

If adapter behavior deviates from expectations—such as unexpected outputs, schema mismatches, or undeclared interactions—these events are intended to be observable and attributable within the system's logging and execution records.

The architecture favors visibility over prevention.

7. Human Authority and Manual Intervention

The system is designed so that human intervention can supersede automated outcomes.

Manual overrides:

- may occur at any stage,

- are treated as explicit state transitions,
- and are recorded as part of execution history.

Automation is considered assistive, not authoritative.

8. Determinism Boundary

Deterministic behavior is an architectural objective, applied to:

- capsule construction,
- execution context generation,
- adapter execution given identical inputs and versions.

Learning, optimization, or adaptation mechanisms—if used—are confined to development or configuration phases and are not intended to modify runtime execution paths.

9. Vault and Trust Boundary Intent

Sensitive or internal data may participate in workflows only through explicit provisioning by the orchestrator.

Design intent includes:

- adapters operating only on provided execution context data,
- vault access being policy-driven and role-aware,
- data scope being minimized and purpose-bound.

This model aims to preserve clarity of responsibility rather than asserting absolute isolation.

10. Hardware Attestation as an Optional Trust Signal

Hardware-based attestation may be used as an optional trust signal to indicate that orchestrator or adapter components are running expected software configurations.

Attestation:

- contributes to trust assessment,
- does not alter execution semantics by itself,
- and is reflected as metadata rather than as a prerequisite.

Absence or failure of attestation is treated as a visible trust-state change, not as a system error.

11. Jurisdictional and Regulatory Positioning

The architecture is designed to be jurisdiction-agnostic.

Regulatory alignment is achieved through:

- explicit policy configuration,
- role and approval modeling,
- and execution transparency,

rather than through built-in legal assumptions or compliance claims.

12. Distinction from Autonomous Agent Systems

The system is not positioned as an autonomous agent framework.

It does not:

- pursue independent goals,
- generate plans outside explicit configuration,
- or alter its operational boundaries dynamically.

All actions remain bounded by declared schemas, policies, and execution contexts.

13. Completion Criterion (Design Perspective)

From a design perspective, the system is considered *sufficiently specified* when:

An implementer can follow the architecture without inferring undocumented intent, and an operator can observe how and why a given execution path occurred.

This criterion reflects clarity of design, not guarantees of outcome.

14. Closing Remarks

This architecture prioritizes explicit boundaries, observable behavior, and controlled extensibility over completeness claims or universal guarantees. It is intended as a foundation that can be incrementally hardened, certified, or extended as organizational maturity and regulatory requirements evolve.

Capsule-Based P2P Communication Networks for Safe Internal and External Automation

1. Threat Model: Why Networks Fall in Practice

Enterprise networks are rarely compromised because a single machine is exploited. They fail because communication channels act as execution vectors.

In real-world incidents, the dominant propagation paths are:

- email links and attachments
- chat messages containing URLs or documents

- collaboration tools with rich previews and file sharing
- user-driven browsing on internal workstations

Once execution occurs on one endpoint, malware spreads laterally via credentials, shared resources, and repeated user interaction. Traditional defenses attempt to *detect* malicious content, but leave the underlying execution model unchanged.

The core problem is architectural:

Communication is treated as data, but behaves like code. It allows blind code execution.

2. Architectural Principle

This system removes execution from communication entirely.

- Internal communication is reduced to capsule-based text and structured artifacts.
- External interaction is never executed on internal endpoints.
- All execution is explicitly offloaded to isolated external devices.
- Policies are enforced as signed data, not UI state.

The result is a communication network that cannot act as a malware propagation layer.

3. Internal Capsule-Based P2P Network

Internal communication uses a peer-to-peer capsule channel, operating over local networks or the public internet.

Properties

- end-to-end encrypted
- identity-bound
- capsule-only payloads
- deterministic parsing
- attachment recreation, originals outside the network
- no rendering engines
- no cloud dependency

The channel transports:

- text capsules
- intent capsules

- policy capsules
- acknowledgments

Transport is explicitly untrusted. Capsules are verified cryptographically before any processing.

Post-Quantum Security

The P2P channel supports mandatory hybrid encryption combining classical cryptography with post-quantum algorithms.

Hybrid operation ensures that:

- new post-quantum key exchange algorithms can be introduced, tested, and rotated without security regression
- long-term confidentiality is preserved even if future cryptanalytic breaks occur
- cryptographic agility is preserved while evaluating and rotating post-quantum algorithms without coupling the capsule model to legacy execution semantics

There is no plaintext or downgrade mode. Trust is derived from cryptographic identity and policy enforcement, not infrastructure.

4. Link Handling: Default External Execution, Optional Admin-Approved Local Open

Links may appear as normal links in the UI. By default, a link click never opens locally. Instead, the click is converted into an intent and executed on an isolated external device. With p2p a scan of a WR Code to open the link in the isolated tablet outside the network is not necessary. The transport of capsules is transport agnostic. If a beap capsule arrives as email or internal network is irrelevant and in both cases safe.

Click Semantics (Default)

A click generates a Link-Intent Capsule, not local navigation.

The capsule contains:

- canonicalized URL
- requesting identity
- target execution device
- nonce and TTL
- cryptographic signature

This capsule is routed to an external execution device (tablet/workstation). Internal endpoints do not fetch or render external content unless an explicit admin-approved whitelist policy permits local opening for that destination.

5. Whitelisting as Signed Policy

Whitelisting is implemented as admin-signed policy capsules, not as client-side configuration.

Technical Characteristics

- canonical domain representation (eTLD+1 / host-level)
- HTTPS-only enforcement
- redirect boundary enforcement
- optional IDN and shortener exclusion
- optional expiration and re-approval

Enforcement Model

- whitelisting is intended only for frequently used, well-established trusted services in an enterprise environment
- whitelist changes require admin approval and are distributed as admin-signed policy capsules
- endpoints apply the whitelist policy deterministically (no heuristics, no previews, no fallback)

Effect

If a destination is whitelisted (admin-approved):

- the URL is allowed to open inside the network on internal workstations
- the same policy constraints still apply (HTTPS-only, redirect boundaries, optional IDN/shortener restrictions)

If a destination is not whitelisted:

- the click is converted into a Link-Intent Capsule and executed on the external device
- execution may be blocked or require explicit approval, depending on policy

6. External Execution and Sanitized Re-Entry

The external tablet or workstation is a controlled execution environment:

- browsing
- authentication
- document handling
- automation

From there, results can be returned to the network only as sanitized capsules.

Example:

- a Word document arrived and is opened externally
- content is extracted and capsulated
- a clean text or structured representation is generated
- the result is returned as a capsule

Original executable artifacts never re-enter the internal network.

7. Security Outcome

This architecture does not attempt to secure heterogeneous endpoints. Instead, it ensures that communication and orchestration cannot amplify compromise.

Structurally Prevented

- link-based malware spread
- chat-driven infection chains
- attachment propagation
- credential theft via internal browsers

Intentionally Out of Scope

- OS-level compromise
- supply-chain malware
- server or directory attacks

Compromised machines remain compromised — but cannot naturally infect others through communication.within the orchestrator

8. Conclusion

Most security systems attempt to make unsafe communication safer. This architecture makes unsafe communication non-executable by design.

By separating intent from execution and isolating all external interaction, it enables cloud-like collaboration and automation while preventing communication-driven lateral movement — without cloud trust, and with post-quantum-ready security.

Administrative actions are classified as security-critical operations and therefore require mandatory multi-factor authentication (2FA) to preserve network integrity under host compromise assumptions. Beyond administrative scope, policy capsules may optionally declare a 2FA requirement as a step-up authorization condition for high-risk or irreversible actions. Where execution is account-bound, 2FA

enforcement becomes an identity-layer control that can be applied globally or per role, aligning authorization strength with the security impact of policy changes and governed actions. In the event of administrative credential or second-factor loss, recovery is achieved through controlled reinitialization of the orchestrator and restoration from verified backups, followed by renewed account binding and 2FA enrollment, ensuring continuity without propagating compromise across the network.

WRChat: Capsule-Based Collaboration, Perception, and Automation in Secure Enterprise Networks

WRChat is implemented as a specialized subclass of the WR communication layer and functions as the primary gateway between internal and external automation domains. It provides a controlled interaction surface where capsules can be sent, received, and exchanged via peer-to-peer or group-based interactions, while enforcing strict role-, time-, and policy-based capability boundaries.

Although exposed in the user interface as a chat-like environment, WRChat is not a messaging system and not a simple capsule builder. It represents a capability-gated control plane that mediates which forms of communication, perception, and automation are permitted between actors, systems, and organizational trust zones. All interactions are strictly realized through BEAP capsules and

evaluated locally under receiver-side policy, with no implicit trust granted by transport or session state.

Importantly, this model applies equally to external interactions and to local system control. WRChat also serves as the policy-governed interface for issuing control and orchestration capsules to the local orchestrator itself. Local actions—such as artifact ingestion, recording of voice or video, triggering automation, or requesting playback—are expressed using the same capsule semantics and evaluated by the same policy engine as external requests. This avoids privileged local bypass paths and ensures consistent capability enforcement, auditing, and governance across all interaction surfaces.

This capsule-first gateway design eliminates the traditional separation between chat systems, file exchange, remote access tools, and automation frameworks. Human communication, semantic instructions, media perception, and automation triggers are unified into a single cryptographically constrained interaction model, while the effective capabilities of each participant and component are dynamically constrained by policy.

Capsule-Native Communication, Capability Gating, and Automation

Every WRChat interaction corresponds to a signed, policy-bound capsule with an explicit intent. Capsules may represent human-readable messages, structured semantic payloads, automation

triggers, media descriptors, artifact references, or local control requests. The mere receipt or creation of a capsule does not imply that its full semantics may be exercised.

WRChat enforces a capability-gated execution model: each capsule is evaluated against a dynamically computed capability set derived from role, time constraints, trust relationship, hardware attestation status, and administrative policy. Capabilities may be temporarily granted, revoked, or restricted in scope (e.g., view-only, trigger-only, no-forward, no-replay).

This capability model applies uniformly to external capsules and to capsules targeting the local orchestrator. By expressing local operations—such as artifact ingestion, recording, automation execution, or media playback—as capsules, the system ensures that even local actions are subject to explicit policy decisions. This design prevents accidental or implicit privilege escalation and enables a consistent audit trail for all control and automation events.

Remote automation is therefore not achieved through exposed APIs, remote shells, or persistent sessions, but through explicit capsule transmission combined with local capability evaluation. Automation always executes locally and deterministically, and only when the receiving system's policy permits the requested intent within the current capability window. This model enables fine-grained external collaboration and automation while preserving strict internal control and auditability.

Provenance-Bound Artifact Handling and Media Consumption

A core design principle of WRChat is that media handling is governed by provenance rather than format. The system distinguishes between different artifact origin classes and applies strict consumption rules accordingly.

Artifacts that are generated entirely within the WR orchestrator stack—including voice memos, videos, display captures, robot camera feeds, AR glasses views, and diagnostic recordings—are considered orchestrator-generated artifacts. Their full creation and processing chain is controlled by verified software components running on hardware-attested devices. Such artifacts may be consumed inside secure enterprise networks, but only through attested decoders and only when explicitly permitted by role- and policy-based authorization. Decryption is performed using post-quantum-ready cryptographic mechanisms and bound to policy-compliant execution contexts. Quantum adversaries gain no practical capability to recover session keys or decrypt content.

By contrast, artifacts ingested from the local environment or originating externally are never opened as original binaries within the network. These artifacts are encapsulated and processed exclusively through secure rebuilds, such as rasterized previews, extracted text, normalized representations, or semantic summaries. Original binaries remain sealed and are accessible only outside the network if explicitly permitted by organizational policy.

This strict provenance-based distinction allows rich internal media interaction without reintroducing the risks associated with arbitrary file uploads or external content consumption.

Attested Perception for Human–AI–Machine Collaboration

As a gateway between internal and external automation domains, WRChat supports attested perception streams from humans, machines, and physical systems while maintaining strict separation between trust zones. Peer-to-peer perception streams constitute a special interaction class and are not encapsulated in the same manner as standard BEAP capsules. Live or time-shifted views from robots, industrial cameras, machinery displays, AR glasses, and other sensor-equipped systems are established through dedicated, policy-gated streaming channels, while capsule semantics are used to negotiate permissions, scope, and derived outputs (e.g., transcripts, events, or automation triggers) rather than to transport the raw stream itself.

Such perception capabilities are not globally available features but explicitly granted capabilities. Policies may restrict perception to specific roles, time windows, operational contexts, or collaboration scopes. For example, a remote partner may be granted temporary, view-only access to a robot camera stream for diagnostic purposes, while automation triggers derived from the same stream remain disabled.

When perception is permitted inside the network, both the producing and consuming endpoints must be hardware-attested and operate within verified software stacks. Derived semantic outputs—

such as transcripts, detected events, object recognition results, or structured automation triggers—may be persisted as capsules, while original media remains governed by provenance, capability, and policy constraints. This enables secure human–AI–machine collaboration without exposing raw device control or unrestricted media distribution.

Semantic Triggers and Policy-Driven Interpretation

Capsules exchanged via WRChat may contain structured semantic markers or protocol tags that are interpreted locally by the receiver. These markers can trigger automation workflows, data transformations, or AI-assisted reasoning according to receiver-defined policy. Interpretation and execution are always local; no remote entity can directly invoke logic or commands on another system. Policy capsules can Policy capsules may grant non-transferable, scope-limited privileges to identity-bound orchestrators within the same trust domain and are executable only after successful multi-factor authentication, such as a hardware security key combined with authenticated user credentials.

This design allows collaboration-driven automation, including voice- or text-triggered workflows, without weakening system boundaries or introducing implicit execution privileges.

Metadata Minimization and Infrastructure Independence

WRChat does not rely on centralized servers, cloud-based control planes, or persistent metadata stores. There is no server-side indexing of conversations, participants, media, or automation flows. All communication occurs over encrypted, hardware-attested peer-to-peer channels. BEAP capsules act as the primary transport abstraction for messages, automation, and artifact exchange, while peer-to-peer livestreams form a distinct, high-assurance interaction class that does not follow the capsule transport model but shares common security attributes such as hardware attestation, policy gating, and cryptographically bound session establishment.

The absence of cloud infrastructure and server-side metadata accumulation significantly reduces the attack surface and eliminates common enterprise risks related to third-party observability, data residency, and metadata leakage. At the same time, the architecture remains compatible with high-security and regulated environments.

Post-Quantum-Ready Peer-to-Peer Transport

All capsule exchange within WRChat operates over encrypted peer-to-peer channels using post-quantum-ready cryptographic schemes. Encryption, authentication, and policy enforcement are bound directly to capsule identity and hardware attestation, rather than to network topology or centralized trust anchors. This ensures long-term confidentiality and integrity even under evolving cryptographic threat models.

Identity- and Envelope-Bound Enforcement of Opening Conditions

To prevent policy circumvention through uncontrolled software or alternative runtimes, WRChat deliberately separates *semantic policy* from *enforceable opening conditions*. Capsules describe what is intended to happen, while the enclosing envelope defines under which verifiable conditions any opening, decryption, or execution is technically possible.

Each capsule is wrapped in an envelope whose identity root (IdentityRoot) cryptographically binds the capsule payload, policy descriptors, recipient binding, required orchestrator properties, attestation claims, anti-downgrade constraints, and key-release mode into a single deterministic hash. Any modification—semantic or structural—changes this root and invalidates signature verification or key release.

Crucially, content decryption keys are never made available based on capsule policy alone. Keys are released only when the receiving orchestrator proves compliance with the envelope's bound requirements, either through sealed-key unwrapping inside an attested runtime or via policy-controlled remote key release. As a result, capsules may be copied or observed, but cannot be opened or executed outside a controlled, attested orchestrator environment.

This envelope-bound enforcement ensures fail-closed behavior: without a valid opening proof that satisfies identity, attestation, and policy constraints, no decryption, replay, or export operation is possible. In this way, WRChat structurally prevents policy bypass and ties authority not to application behavior, but to cryptographically verifiable execution conditions.

BEAP Capsule Extension for Robotics and Autonomous Agents

The previously described WR Code robotics framework—comprising hierarchical roof and leaf WR Codes, Augmented Overlay annotations, vector-based semantic retrieval, and simulation-capable digital twins—can be extended directly through the BEAP Capsule model. This addition enables robots, operators, and AI agents to exchange structured task directives, contextual metadata, and environmental semantics through a single, verifiable, transport-agnostic channel.

1. Capsule-Based Environment Initialization (No Scanning Required)

Where earlier sections describe physical WR Code scanning as the mechanism for environment discovery, BEAP-mediated robotic workflows remove this requirement. A BEAP Capsule may contain:

- roof WR Code identifiers for entire environments,
- leaf WR Code identifiers for devices or control elements (Augmented Overlay) and item specific training data (similar to embeddings for llms but for robotic training data),
- robot-specific WR Codes for manipulators, tooling, or safety envelopes,
- version hashes and WRStamp selectors, and
- references to canonical templates hosted at wrcode.org.

Upon receipt, the robot resolves these identifiers and loads the relevant spatial, procedural, and overlay definitions from its WR Code registry. The registry typically synchronizes with wrcode.org, which provides authoritative, tamper-proof template definitions. For low-latency or offline deployments, robots and orchestrators can cache WR Code templates locally; if no update is available, all overlay, metadata, and 3D environment structures are retrieved from local storage. This enables rapid environment initialization without visual scanning and without reliance on proprietary APIs.

2. Unified Semantic Layer for Machines, Overlays, and Control Elements

As described earlier in the paper, the Augmented Overlay framework applies not only to the orchestrator UI but also directly to machines, control elements, and spatial environments. BEAP Capsules reference these overlays so that robots can interpret:

- machine states and diagnostics,
- procedural guides and safety constraints,
- visual affordances (buttons, arrows, trajectories),
- manipulation instructions,
- expected object zones, and

- device-specific operational logic.

Humans and robots therefore interpret the same metadata through different perceptual channels—AR for humans, semantic lookup and perception alignment for robots—ensuring consistent understanding of the environment.

3. Task and Reasoning Coordination via BEAP Messaging

Robots may receive BEAP Capsules that encode:

- high-level goals,
- stepwise task procedures,
- requested data,
- reasoning triggers,
- safety rules,
- augmented overlay updates, or
- device-level manipulation steps.

Capsule responses from the robot can include:

- sensor readings,
- state changes,
- alignment confirmations,
- low-level execution logs,
- vector-based semantic matches,
- trajectory evaluations, and
- follow-up questions.

This creates a bidirectional reasoning channel between robot and orchestrator. Robots operate autonomously but can escalate ambiguities or safety-relevant uncertainties to the operator via Capsule-based queries.

4. Operator Interaction: Live Chat, Monitoring, and Analytical Oversight

The BEAP orchestrator enables operators to supervise and refine robotic behavior through structured p2p Capsule messaging. Operators may:

- chat with the robot in natural language,
- supply missing context or parameters,

- override or approve actions based on policy,
- provide requested data fields, and
- adjust reasoning flows.
- Augmented VR/AR overlays allow contextual modification of WRCode-registered machines and control elements. Capsules may be sent directly from remote operators to VR/AR devices, where context can be reviewed and further adjusted in the 3D overlay or via AR glasses, then securely propagated to robots—supporting coordinated updates across distributed human–robot teams. At the same time, the orchestrator provides real-time monitoring, including:
 - camera feeds,
 - overlay-aligned viewpoint renderings,
 - manipulation trajectories,
 - environment maps,
 - semantic reasoning traces, and
 - safety envelope visualizations.

A multi-panel display grid allows operators to compare simulation, actual robot motion, environment overlays, and debugging information simultaneously.

5. Simulation, Training, and Real-World Execution

Because WR Code environments are represented in open formats (USD, glTF), they can be used seamlessly in simulation environments such as NVIDIA Omniverse, Unity, or Unreal. Robots can be trained using:

- the same WR Codes,
- the same Augmented Overlays, and
- the same manipulation metadata

that appear in the physical workspace. BEAP Capsules permit switching between simulation and real-world execution by referencing the same WR Code hierarchy, enabling consistent task transfer with minimal recalibration.

6. A Cohesive Framework for Robotic Control

With BEAP Capsules integrated into the robotics architecture, the orchestrator becomes a complete control and reasoning environment, capable of:

- initializing environments through WR Code references,

- coordinating multi-agent reasoning,
- performing semantic retrieval on vector databases,
- visualizing robot behavior,
- exchanging interactive BEAP messages with robots,
- enforcing safety and policy constraints,
- synchronizing digital twin and physical execution, and
- maintaining WRStamp-based integrity across tasks.

Robots, operators, AR systems, and AI agents therefore share the same verifiable, structured semantic substrate—allowing transparent, explainable, and interoperable robotic workflows.

WRStamp for Verifiable AI-Generated Media

A Trust Anchor for the Synthetic Content Era

As AI-generated videos, cloned voices, and synthetic media become increasingly realistic, the risk of misinformation and impersonation grows rapidly. From deepfake press releases to fraudulent voice

messages, today's information systems need a verifiable trust layer that works across formats and platforms.

WRStamp introduces a cryptographically verifiable framework that allows any publisher to prove authorship, ensure content integrity, and enable independent validation—whether for videos, voice messages, or phone calls.

What WRStamp Enables

Each WRStamped asset (text, video, audio, etc.) includes:

- A deterministic cryptographic content hash
- A signed metadata template (e.g., purpose, timestamp, publisher identity)
- A publisher keypair, verified via wrcode.org
- A mandatory anchoring transaction on a public blockchain (e.g., Solana, Optimism, or IOTA)

This enables third parties to verify that:

- The content has not been altered
- It originates from a registered, verified entity
- The publication context is auditable and anchored immutably

 Verifiable Phone Calls — Secured by WR CODE

WRStamp enables cryptographically verifiable phone calls through a secure, source-available background layer—without changing how you place or receive calls.

 1. One-Time Trust Handshake

A handshake is initiated by a verified WRCode account holder logged into the secure layer of WR CODE. The recipient—also logged into the secure layer on their own device—receives the handshake (e.g. via Signal or WhatsApp) and must actively confirm it. No scanning is required.

 Once confirmed, a peer-to-peer (P2P) connection is established between the two systems. This connection is used to:

- Verify the identity and software integrity of both parties
- Ensure device- and session-bound authenticity
- Perform a secondary validation step resistant to man-in-the-middle attacks

 Only if this P2P handshake verification succeeds, the trust relationship is anchored immutably on Solana, establishing a cryptographic record of authenticity. This two-layer design ensures that:

- No fake or manipulated data can be transmitted undetected

- Only verified systems can interact within the trusted context
- Trust remains revocable and auditable at all times

📞 In real-time communication scenarios (e.g. phone calls), where pre-hashing is not feasible, a live "ping-pong" exchange over the secure P2P channel ensures continued verification. If internet fails, a fallback validation via inaudible GSM tone can keep the handshake alive.

🛡️ In an era of deepfakes and synthetic impersonation, this layered approach provides tamper-proof authenticity—not just once, but throughout the full lifecycle of each interaction.

🔒 No personal data is shared. Only hashed values and zero-knowledge proofs are exchanged.

✓ 2. Call Authentication via Background Orchestrator

Once the handshake is confirmed and WR CODE is running in the background on both sides, all future calls between trusted devices are automatically validated:

- When a call is initiated, a one-time token is generated and sent via Ping Pong
- The recipient's orchestrator checks:
 - Is the device part of the approved trust set?
 - Is the token valid and untampered?

If everything matches, the call is silently confirmed.

If not, the orchestrator displays a red warning, flagging the call as untrusted or suspicious.

3. Privacy-Respecting and Source-available

- WR CODE is fully Source-available, with built-in runtime integrity checks
- All trust logic relies on zero-knowledge cryptographic proofs

4. Trust That Evolves

- Handshakes can be revoked, expire, or persist
- Device sets can be updated—automatically rehashed and revalidated

Optional Post-call Confirmation

"This call has been verified under session WR-0xABCD123 and cryptographically anchored."

Result:

Your phone works as usual—WR CODE just runs in the background to ensure every call is verifiable, tamper-resistant, and private.

No behavior change. No metadata leakage. Just proof of trust, built in.

❖ In many parts of the world, a knock at the door is more than an inconvenience — it's a calculated risk. Criminals posing as police officers, utility workers, or delivery drivers have used this simple act to commit theft, fraud, extortion, assault, and even rape. In countries with fragile institutions, these impersonation tactics are brutally effective. But even in modern, well-regulated societies, the danger persists: trust is still granted too easily, based on uniforms, badges, or tone of voice.

WRCode introduces a new standard — one rooted not in appearance, but in cryptographic truth. With it, individuals and organizations can verify identities at the doorstep, gate, or entry point using decentralized, tamper-proof handshakes. Critically, this validation works even without an active internet connection, enabling security in the places and moments where it's most urgently needed.

Offline Identity Matching for Real-World Encounters

While WRCode supports full offline verification, it is primarily designed to operate with active internet connectivity. Online verification ensures real-time integrity checks, anchor lookups, and trust updates across dynamic contexts — making it the most reliable mode of operation in both civilian and institutional settings.

Offline capability serves as a fallback — crucial for areas with limited infrastructure, emergencies, or network outages — but it complements rather than replaces the online validation model.

WRCode-enabled devices can securely recognize and validate known individuals or service providers in proximity — without revealing personal information or relying on central infrastructure. This is enabled by WR CODE, a secure runtime layer where each software installation is cryptographically anchored and tamper-proof.

Each WR CODE instance is uniquely bound to a Merkle root during installation and anchored on-chain (e.g., Solana for speed, IOTA for audit). Only verified, unaltered clients can participate in handshakes.

Two Types of WRHandshakes

1. Peer-to-Website (P2Website) — user validates a service provider through a WRCode-enabled website.
2. Peer-to-Peer (P2P) — two devices establish a direct, cryptographically anchored relationship.

Each handshake requires a one-time online verification, after which both parties store a cryptographically bound trust token:

- Account hash
- Device fingerprint (via ZKP)
- Timestamp and scope

- Software group identity (bound to installation)

Offline Use Cases

Once established, trust tokens enable secure offline recognition:

- A friend, employee, or colleague confirming arrival
- A technician or courier proving legitimacy
- A uniformed agent triggering a verified notification

The recipient's device detects local handshake signals (Bluetooth, Wi-Fi Direct, ultrasonic), verifies the token, and compares a locally stored hash from its PII Vault against the visitor's transmitted request hash. No PII is ever shared — the match only succeeds if both sides independently possess matching contextual information, such as a delivery address, appointment name, or recipient alias. This information is naturally known to legitimate visitors: a police officer executing a search warrant knows the target address; a courier or technician has the delivery or service destination; a friend knows your name or secret and residence. The pre-established handshake ensures that this data can be cryptographically validated without exposing it. If the hash comparison is successful, the device displays a notification such as "Amazon Delivery" or "Plumber has arrived."

All validation happens locally and privately — with no raw PII exchange.

Structured Matching with PII Vault

In scheduled scenarios (e.g., delivery routes), agents can preload hashed contact data. If the resident's device has a long list of expected visits — such as multiple deliveries or appointments — it dynamically sorts matches to the top when a valid handshake is detected, making identification fast and intuitive:

- "Amazon Delivery, Hans Meissner, Rainystret17"

On approach, the agent's device computes a request hash. The resident's device compares this against its own stored PII hash. If matched, a "Confirm Visit" button appears. Once tapped, a notification is sent.

Security Properties

- No PII leakage: ZK-validated hashes are used
- No internet required: validation can also be done offline
- No spoofing or replay: tokens are signed and session-scoped
- No forced alerts: confirmation required for notification

⌚ Real-World Visit Logging (Optional)

Once a WRHandshake is successfully verified, the actual event of a real-world visit or physical interaction can be optionally logged on IOTA using zero-knowledge proofs. This allows individuals or services to retain tamper-proof records of time and intent — without revealing personal data — ideal for first dates, deliveries, access control, or sensitive encounters.

Building a Trust-First Digital Layer

The vision extends further: WRCode is not limited to mobile or handheld use. Future integrations may include WRCode-enabled intercom systems, smart doorbells, and access cameras. These devices could verify a visitor's cryptographic identity while video is already available — ensuring that visual confirmation is accompanied by a cryptographic trust signal— allowing residents to confirm who is at the door without relying on unverified visuals. With WRCode embedded in residential infrastructure, trust becomes a built-in layer of physical access — not a manual guesswork process.

WRCode enables lightweight, privacy-first identity verification for personal, residential, and institutional use. It supports a gradual shift to secure, offline-capable interactions that are verifiable by design — protecting against impersonation, deception, and unauthorized access in both everyday and high-risk environments.

Only previously validated and downloaded AI automation templates, code libraries and tools—originating from an earlier online-authenticated handshake—can be used offline. No new automations or actions are executed while offline.

WRHandshakes can also be initiated through a tamper-proof, source-available secure element embedded in websites, allowing companies and services to establish trust with users even if only one party runs the full WR CODE secure layer.



How WRCode-Enabled Software Establishes Tamper-Proof Trust on Trusted Websites

To prevent handshake-capable software (e.g., WRHandshakes) embedded in websites from being spoofed, duplicated, or modified without detection, WRCode enforces a layered, cryptographically anchored installation protocol. Each installation is permanently bound to its original deployment context and verifiable at runtime via Solana anchoring. This design maintains strict control over authenticity, location, and integrity. Note: These same security measures apply to *any* WR CODE-based installation capable of establishing WRHandshakes — including browser-based orchestrators or endpoint-integrated runtimes. All trust assumptions are enforced through identical cryptographic procedures, regardless of context.



Layer 1 – One-Time Secure Installation with Cryptographic Snapshot

All installations are executed within the WR CODE secure layer under strict conditions. During installation, the software generates a cryptographic snapshot that binds the deployment to:

 Server Environment (Remote)

- Hardware fingerprint (TPM, CPU ID, or other stable identifiers)
- Public IP and ASN-based geolocation
- Full DNS and domain record structure
- Coordinated UTC installation timestamp

 Installer Device (Local)

- Active WRCode account and authenticated session
- Installer device fingerprint (validated via local-only zero-knowledge proof)
- Installer's geolocation and timestamp at the time of session

 Software & Trust Metadata

- Full software code hash (immutable; shipping is single-use)
- Unique software ID tied to the current installation session

- TLS certificate fingerprint — included at install time as part of the Merkle tree, strengthening first-time trust anchoring

All elements are committed to a Merkle tree, whose root hash serves as the immutable fingerprint of the installation. This root is anchored on the Solana blockchain, providing public, tamper-proof proof of origin.

Layer 2 – Runtime Integrity Verification with Solana Anchoring

During each WRHandshake attempt, the embedded software performs a real-time integrity check by recomputing and validating the critical runtime parameters:

Runtime-Validated Parameters:

- Current DNS configuration and domain
- Server hardware fingerprint
- Deployed software hash
- Validation of the anchored Merkle root against the Solana blockchain

This ensures the software is still running in its original environment, has not been relocated or altered, and has not been revoked.

🔗 Peer-to-Peer (P2P) Handshake Protection

For decentralized p2p contexts (e.g. device-to-device handshakes), WRCode enforces an additional challenge–response mechanism that prevents:

- Man-in-the-middle attacks, through locally scoped zero-knowledge proof-based validation
- Spoofing, by requiring both peers to verify cryptographic bindings established at installation time
- Replay attacks, by binding handshake tokens to ephemeral session parameters and timestamped commitments

This layer allows WRHandshakes to retain the same tamper-proof and origin-bound properties even in the absence of a server-bound validation context.

🚫 Prevention of Spoofing, Duplication, and Relocation

Each installation is:

- Cryptographically bound to a specific account, environment, location, and timestamp

- Non-transferable — reinstalling or cloning invalidates the runtime checks
- Uniquely anchored — any mismatch with the original Merkle root causes rejection

Revocation and Lifecycle Controls

- WRCode administrators can revoke any installation at any time through an online interface
- Revoked installations are blacklisted on-chain
- Runtime checks query revocation status in real time before executing handshakes

Security Summary

| Attack Vector | Mitigation |
|---------------------|--|
| Software tampering | Blocked by Merkle-rooted code hash |
| Server relocation | Detected via hardware fingerprint mismatch |
| Domain Spoofing | Detected by domain and DNS revalidation |
| Certificate changes | Tolerated due to runtime exclusion of TLS |

| Attack Vector | Mitigation |
|-----------------------|---|
| Replay or duplication | Blocked by time, location, and account binding |
| Silent compromise | Fails real-time Solana verification or revocation check |
| MITM in p2p context | Blocked by challenge -response and ZKP -bound tokens |

Conclusion

This multi-layered architecture ensures that WRCode and WR CODE-based installations:

- Are immutably linked to their point of origin
- Cannot be cloned, spoofed, or silently modified
- Perform continuous self-verification before executing any sensitive operation
- Allow full administrative control over lifecycle and revocation

Trust is not assumed — it is enforced cryptographically, in real time, across all handshake-capable environments.

Revoking WRCode Handshakes via Passive Anchoring

Overview

WRCode handshakes establish trusted relationships between two parties — typically between a user and another user a plugin, service, machine or another device. Like WRToken licenses, these handshakes are cryptographically signed and then anchored as part of the pairing process. The signature ensures origin integrity, while anchoring binds the trust relationship to a public audit trail.

To allow revocation of these relationships (e.g., a user unpairing a device, or rescinding a handshake with a service), WRCode supports an optional, passive anchoring mechanism using public ledgers such as IOTA and Solana.

This allows one or both parties to publish a revocation marker. This allows trust to be revoked in a verifiable way, without relying on remote approval, smart contracts, or centralized DRM-like control patterns.

How Handshake Revocation Works

WRCode handshakes are always anchored during the initial pairing process as part of the integrity and trust model. However, revocation actions themselves can be initiated locally by each authorized party — without needing a new handshake or coordination with the original counterparty. This ensures rapid, self-sovereign invalidation when needed.

If public auditability is required — for example, in system-critical infrastructure, regulated environments, or simply for peace of mind — a revocation hash may optionally be anchored to a distributed ledger like IOTA or Solana. While technically unnecessary for local enforcement, anchoring can provide additional external assurance. This anchoring is not mandatory and serves purely as a public proof-of-revocation when transparency is needed.

A unique strength of WRCode is that revoking a handshake does not require another handshake.

Any authorized party — typically the originator of the handshake — can unilaterally publish a revocation hash. This makes it possible to revoke trust relationships even if the original counterparty is unreachable, offline, or compromised.

1. A WRHandshake is established, resulting in a signed, shared handshake hash (e.g., derived from both parties' IDs + timestamp + domain).
2. If one party wants to revoke it, they publish the hash of the handshake ID to a public ledger under a standard tag, e.g.:
 - WRREVOKE:HANDSHAKE
3. The WRCode holder can optionally verify the revocation status by checking the anchored hash on a public ledger

All enforcement remains local and WRStamped — no live queries or smart contract logic.

Legal and Privacy Considerations

WRCode revocation anchors are passive markers only:

- No content of the handshake is published — only a cryptographic hash
- No dynamic approval is required to enforce revocation

This ensures full GDPR compatibility, avoids behavioral tracking, and stays clear of patent-sensitive smart licensing techniques.

Cross-Platform Integration for Other Media

WRStamp supports platform-independent integration across YouTube, websites, messaging platforms, and more:

Embedded metadata

- Video/audio: XMP or stream-level metadata (e.g., wrstamp_hash, publisher_id)
- Web: JSON-LD in <head>, or .well-known/wrcode.json publisher entry

Visible trust anchors

- QR overlay (WRCode) in video or images
- Plain-text verification links:
"✓ Verified by WRStamp – View authenticity"

Public publisher registry

- Registered publishers include:
 - Public Key fingerprint
 - Verified entity info (e.g., organization, legal ID)
 - Trust-level badges
 - Public signature feed (signed content, hashes, timestamps)

Source-available verification

- WRGuard (CLI, browser plugin, or server tool) validates:
 - Hash signature match
 - Anchor presence
 - Publisher authenticity

🚫 Can WRStamped Links Be Abused?

A WRStamp verification link is publicly viewable, but does not grant impersonation capabilities .

By using:

- Recipient-bound hashes (e.g., tied to phone number)
- Expiry times
- Signed session context
- Tamper-proof anchors

...WRStamp ensures that trust is context-specific . Even if a link is shared or intercepted, it cannot be reused convincingly without matching the original content, voice, and metadata.

🛡 If the call or content doesn't match the WRStamp — it's not authentic.

🌐 Limitations and Responsible Framing

WRStamp is not designed to detect AI-generated content or prevent malicious generation. Instead, it provides a verifiable origin and context layer , enabling humans and machines to distinguish trustworthy from unverifiable digital content.

The current implementation requires:

- Publisher registration and key management at wrcode.org
- Public anchoring on supported ledgers
- Use of WRGuard or compatible verifier tools

Conclusion

WRStamp brings cryptographic trust to AI-era communication—across video, voice, and phone calls.

With its hybrid approach of signed metadata, blockchain anchoring, and verified publisher identities, WRStamp allows digital content to become not just believable, but provably authentic.

OpenGiraffe



Geographic Origin Verification and Policy Enforcement in WRCode

WRCode introduces a groundbreaking system for cryptographic proof-of-origin that permanently binds every WRStamped object to the country in which it was created. This includes emails, code, automation templates, documents, or any digital asset bearing a WRStamp. The origin is determined using a decentralized verifier swarm, local environment signals, and a zero-knowledge proof scheme. The process is source-available, privacy-preserving, and entirely trustless. Once established, the country-of-origin cannot be changed, removed, or spoofed.

How the Origin Verification Works

The proof-of-origin process is enforced during signup. A delayed, background verification is launched and executed locally with support from a decentralized verifier swarm. This swarm performs time-based and multi-angle verification, aggregating cryptographic signals derived from the user's environment — without exposing personal data. These signals are evaluated using a zero-knowledge protocol that produces a signed proof of geographic origin. This origin proof is permanently bound to the user's WRVault identity and is anchored on a cryptographic ledger for tamper-proof, timestamped validation. Vaults refuse to issue WRCode or WRStamped content without this binding.

Policy Enforcement by Geography

This allows individuals, businesses, and governments to enforce trust policies based on verifiable geography. For example, a company may block any WRStamped email or automation originating from a country it has chosen to blacklist — of its own choice. While recipients can choose to blacklist or whitelist certain countries, the proof-of-origin itself is cryptographically enforced and cannot be tampered with. For most organizations, it makes little or no sense to receive unsolicited offers, code, or proposals from countries to which they have no business connection. These "business" contacts become functionally obsolete once origin verification is enforced. This dramatically reduces irrelevant and unsolicited inquiries, lowering the global volume of digital noise and spam — which also has a positive environmental and operational effect.

The system can also help detect and expose **false claims** of origin. This protects against Spoofing and impersonation at the infrastructure level.

WRCode does not block any user by default. But it ensures that **every** signed object includes a cryptographically verifiable trail of its real origin. This empowers each participant to define their own trust boundaries based on cryptographically verified origin — without relying on intermediaries or external enforcement.

Why This Matters

While WRCode is primarily designed to provide secure, tamper-proof automation and trusted workflow execution, its underlying architecture has far broader implications. The same cryptographic origin binding that secures automation flows can also be applied to digital content such as verified source code, libraries, documents, or proprietary binaries. This opens the door for WRCode to become a universal trust protocol for all software distribution — from source-available packages to licensed enterprise platforms.

Today, proprietary software can be updated silently, and users have no cryptographic way to verify where the code came from or if it has been tampered with. This is not just inefficient — it is dangerous. WordPress, which powers over 40% of all websites globally, is a prime example. Each plugin or theme update has the potential to introduce malicious behavior, with no proof of origin or authenticity attached. In this environment, bad actors operate in silence while users — even technical ones — are left defenseless. The complexity of modern software systems makes it virtually impossible for any individual or team to manually inspect and verify every line of code multiple times a day to detect malicious updates. Without a cryptographic enforcement layer, changes can be introduced silently, without auditability or attribution — and this is exactly how attackers thrive today. Even direct hacks or code injections can be thwarted by WRCode: if malicious code is injected into a WRStamped object, the WRStamp verification will immediately fail, causing the content to be

rejected or halted at execution. The integrity check becomes automatic — if the code changes, the stamp breaks, and the attack is stopped in its tracks.

WRCode gives defenders back control. It enables systems where every piece of content, automation, or executable is tied to a verified, immutable origin. Once this standard is adopted, it will render unverifiable content obsolete. Attacks that today happen in silence — damaging reputation, trust, or infrastructure — will become detectable and preventable at the protocol level. People have been forced to accept that their systems can be silently compromised. WRCode breaks that paradigm. It replaces blind trust with provable, verifiable facts — and introduces a new level of digital self-defense. In today's digital world, malicious software, communication, and even open source code libraries and open source tools can be created, shared, and executed anonymously and globally. There is no enforcement layer that ties digital content to physical jurisdiction. Code hosted on public repositories can be updated silently, from untraceable origins. WordPress sites are hacked daily. Emails are spoofed. Documents are forged. Innocent people, hospitals, small businesses, and even national infrastructure are attacked without any way to determine where the threat came from.

WRCode changes this by introducing a layer of accountability that operates not through surveillance, identity checks, or government cooperation, but through mathematics. Every WRStamped file or message either includes proof-of-origin or is automatically blocked or flagged by recipient Vaults according to policy. Users define what to trust. The protocol enforces it.



WordPress Trustless WRStamped Security



WRCode for WordPress

WordPress is an open source content management system used by over 40% of all websites worldwide. While WRCode's primary focus is secure automation, its mechanisms offer powerful security implications for platforms like WordPress — which are frequently targeted by malicious actors. We aim to enable WRCode-verified WordPress setups, where all core files, themes, and plugins are WRStamped. In this model, any unauthorized modification or silent code injection immediately invalidates the WRStamp, making the change cryptographically detectable. A WRCode WordPress installation would detect any code change or injection immediately, making it extremely difficult for attackers to introduce harmful vectors without being noticed. This would make silent compromises — which currently go unnoticed — a thing of the past.

Such a system could transform WordPress from a widely exploited attack surface into a proof-based, jurisdiction-enforced platform. In the broader vision, WRCode could also be used to secure login workflows for WordPress — but only on installations that are fully WRStamped and compliant with defined integrity standards. This would prevent login scripts or authentication mechanisms from being silently tampered with. Given the scale and influence of WordPress worldwide, this platform clearly qualifies as a priority candidate for secure-by-design implementation under the WRCode model. Implementing WRCode on platforms like WordPress demonstrates the broader power of cryptographically verifiable origin — even in systems that were never designed with deep trust enforcement in mind.

 Real-World Impact

This changes everything:

- Developers who offer code in public libraries can be held accountable if their work is misused or corrupted.
- Businesses can verify and filter digital supply chains based on geography, not branding.
- Governments and critical infrastructure providers gain a tool for preventing attacks based on mathematical origin control , not firewalls and guesswork.
- Individuals get the right to say: "I do not want to receive code or messages from unverifiable sources."

Integrity checks are standard in high-risk infrastructure—but almost unheard of in **WordPress** or other open source systems. Open Source should not be the second choice when it comes to security.

WRGuard's mission is to make it the first.

- **WRGuard** brings that missing layer to the open web.
No more blind trust. No more silent breaches.
Just cryptographic proof that what runs is what was intended.
- Because security should be built-in, not bolted on.
And Open Source deserves real defenses.

WRStamped Enforcement Framework

Trust every line — or run nothing. The internet runs on assumption: That plugins are clean. That themes aren't tampered with. That secrets are safe because files "look untouched." Visual polish can be deceiving. Without verifiable identity and code integrity, even the most professional-looking service may be a well-disguised threat. But these assumptions don't hold up — especially on platforms like WordPress, where dynamic code execution, auto-updates, and weak file controls are the norm.

The WRStamped Enforcement Framework replaces that assumption with cryptographic trustless security.

- All executable components beside wp-config.php must reside in WRStamped files. Any deviation halts execution prior to runtime inclusion.
- X If even a single line of running code is not WRStamped — the system halts. This enforcement operates at the PHP application layer using mu-plugins — not UEFI, firmware, or kernel.

Why You Must WRStamp Your Own WordPress Core

In WRGuard-secured WordPress installations, each user is responsible for generating their own WRStamp. Even if the WordPress core comes from a trusted source, the final deployed stack always contains local modifications—making a generic, pre-signed WRStamp insufficient.

To aid transparency and reproducibility, official WRStamp hashes for clean WordPress core files (excluding wp-config.php) are published and verifiable. But they serve only as a reference baseline—not as a substitute for user-specific stamping.

Why Local WRStamping Is Required

Each deployment introduces environment-bound elements that change the hash:

- Moved or encrypted wp-config.php
- Injected vault logic
- Custom security hardening (e.g., file removal, plugin restrictions)
- Additional bootstrap layers like WRGuard

Even with a clean base, the final runtime state must be verifiably sealed by the operator.

What Must Be WRStamped Locally?

-  Core WordPress files (minus wp-config.php)

- Custom or modified plugins
 - All child themes, especially custom ones
 - WRGuard enforcement logic
 - Optional: static assets, custom templates, CLI scripts
-

Tooling: Making Local WRStamping Easy

The WRStamping toolkit will provide:

- CLI and GUI interfaces
- Automatic scanning of your WordPress root
- Exclusion of environment files (wp-config.php, .env, etc.)
- Integrity snapshot creation
- Optional anchoring on IOTA, Solana, or local WORM storage

This makes it simple for non-technical users to generate fully verifiable, locked-down deployments with minimal effort.

Published Reference Hashes

To verify the integrity of the base system, WRCode will publish:

- Official WRStamp hashes for clean WordPress versions
 - Hash lists will exclude wp-config.php and local files
 - Signed manifests for use with CI pipelines or forensic comparison
-

Summary

Every WRStamp reflects the unique, deployed runtime.

Even if your stack starts with verified sources, the final WRStamp must be yours.

Custom child themes, locally built plugins, and all hardened logic must be included.

WRStamping tools will make this process seamless—even for advanced, modular stacks.

 Zero-Exception Integrity

The WRStamped model enforces total file integrity across the entire runtime stack. It doesn't "scan for threats" — it refuses to run unverified code.

| Component | WRStamped Required |
|--------------------------|---|
| Plugins | <input checked="" type="checkbox"/> |
| Themes | <input checked="" type="checkbox"/> |
| SQL execution paths | <input checked="" type="checkbox"/> (via Merkle-root trust chain) |
| Vault/decryption logic | <input checked="" type="checkbox"/> |
| WordPress core overrides | <input checked="" type="checkbox"/> |
| Custom PHP/JS code | <input checked="" type="checkbox"/> |
| Shortcodes, handlers | <input checked="" type="checkbox"/> |

Only the media upload folder is exempt — and only under strict filetype and MIME restrictions (no .php, .js, .sh, .bin).

Secrets Are Bound to Verified Code

Credentials (like DB passwords, API tokens, or SMTP keys) are:

- Stored encrypted, outside of plaintext files like wp-config.php
- Decrypted only at runtime — and only if:
 - The full runtime passes WRStamp verification
 - No file has been added, removed, or altered

 If one WRStamp breaks, the decryption fails — and the system enters lockdown.

There is no fallback, no override, no “admin mode.” Integrity is not optional.

Query Logic: No Rewrite Needed — Just a Verified Roof

WRCode supports inline SQL logic, as long as the execution path is WRStamped.

- The system uses a Merkle-style trust tree to validate all query-relevant files
- If the file or logic that generates or executes a query is not WRStamped → the query is blocked

- No need to rewrite every SQL call — just verify the full path leading to it

A query must live under a cryptographic roof. If the roof leaks, nothing goes through.

Licensing Without eval() — A New Standard for Proprietary Software

Under WRCode, legacy techniques like eval() and runtime code injection are no longer permitted — and no longer necessary.

For decades, plugin vendors used eval() to:

- Inject licensing logic dynamically
- Obfuscate proprietary code
- Load remote conditions or toggles at runtime

These approaches were not only insecure — they became the primary attack vector for malware, piracy workarounds, and supply chain attacks. WRCode ends this era. Licensing and feature control can now be enforced without exposing logic or relying on dynamic execution.

How WRCode Handles Licensing — Without eval()

WRCode supports license scope enforcement without runtime obfuscation or cloud callbacks. License logic is embedded into WRStamped modules using signed, hash-anchored manifests. Execution only proceeds if the verified scope matches the user's WRCode account.

| Need | Legacy Method | WRCode Approach |
|-------------------|--|--|
| Verify license | <code>eval()</code> or dynamic callbacks | Signed WRStamp tied to account hash or device ID |
| Tiered access | Inline PHP logic or remote API | Pre-stamped code bundles with license scope metadata |
| Trial expiry | Obfuscated timestamp logic | Time-bound WRStamp validity window |
| Piracy protection | Obfuscated logic / license server | Immutable, publicly verifiable execution path |

WRCode does not rely on DRM or remote callbacks. Licensing is defined statically at packaging time using anchored metadata and account-bound configuration. Templates are executed only if their license scope matches the verified WRCode account.

The result:

- No runtime tampering
- No secret logic injection
- No trust-by-obscurity
- No reliance on cloud DRM or eval()

License enforcement relies on pre-stamped metadata and account-bound tokens. Optional privacy-

How WRCode Licensing Works

WRCode enables software publishers to license code without relying on runtime injection, obfuscation, or DRM. Instead, publishers generate WRStamped modules: self-contained packages that include their code and a cryptographically signed manifest. This manifest defines the license scope — such as the intended WRCode account, expiration date, and allowed features.

To distribute a license, the publisher issues a lightweight WRToken: a signed JSON file or QR-encoded payload that confirms the recipient is authorized to execute the module. These tokens may be optionally anchored on IOTA or other ledgers for tamper-proof auditability.

At installation or update time, the user's system verifies that:

- The WRStamp and WRToken match
- The token scope includes their WRCode account

- The validity period has not expired

Only then is the module allowed to run. No remote servers, no runtime callbacks, no hidden logic.

Local-Only License Verification (Without Blockchain)

- If a vendor chooses not to anchor the token on a public ledger, WRCode still provides strict local enforcement using static, auditable metadata:
- The WRToken must be signed with the vendor's WRDev key, as declared in the module's WRStamped manifest
- The license scope and identity are fully disclosed in the accompanying WRStamped PDF, which every Plugin, Template, or Template Kit includes
- The license may also be embedded directly in the module payload or the shipped AI templates, allowing compact and portable enforcement
- No injected logic, no obfuscation, and no dynamic resolution: all checks occur at install time and are fully transparent

This approach allows secure use in offline and private settings while preserving auditability and tamper-proof properties, even in the absence of external connectivity.

 Security Without Central Dependency

WRCode does not mandate blockchain use for license enforcement. Vendors may choose:

- Public anchoring (IOTA, Solana) for global verifiability
- Fully offline deployment using verifiable static metadata, WRStamped packaging, and transparent local validation

This model avoids patent conflicts by relying exclusively on open source cryptographic tools, static verification, and transparent, tamper-proof metadata — not behavioral monitoring, runtime callbacks, or proprietary DRM.

 Optional Anchoring of WRCode License Status on IOTA and Solana

Overview

WRCode is built on a simple but powerful principle:

 All license and security enforcement happens locally, using WRStamped logic, without smart contracts, remote callbacks, or runtime code injection.

To enhance transparency, auditability, and verifiability, WRCode optionally supports anchoring license status markers (such as revocation declarations) on public distributed ledgers such as IOTA and Solana.

This article explains how this optional anchoring works, how plugins can verify it safely, and why all critical decisions still happen entirely offline.

What Is Anchored?

Anchoring in WRCode refers to the process of publishing cryptographic hashes of license state metadata (e.g. revocation flags) to a tamper-proof public ledger. This includes:

✗ License Status Anchors

- A hash of a WRAccount or WRToken can be anchored to indicate that a license is no longer active
- WRCode defines standard tag formats like WRREVOKE:<plugin> or WRREVOKE:ALL
- Plugins can periodically check whether the associated account or token appears on a public inactive list

This provides cryptographic verifiability of license status — without introducing remote dependencies.

Where Is It Anchored?

⚡ IOTA (Feeless, Lightweight)

- Uses IOTA's indexation tags to publish WRCode-related hashes
- Ideal for embedded, privacy-sensitive, and open infrastructure deployments
- Resilient, public, and energy-efficient

⚡ Solana (Fast, High Throughput)

- Uses Solana transaction logs or memo fields to embed hashes
- Supports fast publication and large-scale deployment workflows
- Suitable for commercial SaaS plugins and enterprise environments

WRCode supports anchoring to both networks simultaneously for redundancy and neutrality.

How Anchoring Is Used

Anchoring is never required — but when enabled, WRCode plugins may verify anchored data like this:

✗ License Invalidation Verification

1. Plugin hashes its active WRAccount or WRTOKEN
2. Periodically checks whether this hash appears in a known WRREVOKE: ledger anchor
3. If found:
 - Plugin exits or disables features, according to WRStamped local policy
4. If not found:
 - Plugin enters a locally pre-defined grace period, or continues normally based on internal WRStamped logic

All enforcement is embedded in local, signed logic — not defined by the ledger.

Fallback Strategy & Publisher-Defined Logic

Anchoring is designed to be resilient, not controlling.

If neither IOTA nor Solana is reachable, or no anchors are available, plugins fall back to predefined logic embedded in the WRStamped validator. WR CODE operates as a construction kit, giving plugin

developers the freedom to implement their own anchoring mechanisms if desired — including custom ledger integrations (e.g., Optimism or other high-performance L1s).

⌚ Default Behavior

- The plugin continues using only local WRToken + logic
- No network call is required to run
- Status checks are skipped if anchors are unavailable

⏳ Grace Period (Optional)

Publishers can define a grace period, e.g.:

- "Allow 7 days without status anchor check"
- "Continue execution, but warn the user"
- "Disable premium features, apply rate limits, or enforce reduced access — all based on the logic defined by the plugin publisher"

🔧 Business Logic Control

Plugin publishers are free to define their own anchor-dependent rules, such as:

- Required confirmation on IOTA and Solana
- Anchor freshness within X hours
- Tiered execution modes (e.g., allow offline but limit specific features)

All policies are WRStamped and cannot be changed at runtime. However, developers should avoid placing any license logic directly on-chain. Anchors are meant solely for passive verification and must not include executable license enforcement logic, which is a legally sensitive domain and may involve patent-protected techniques.

What Anchoring Is Not

✗ Not Used

Why

Smart contracts

Risk of patent conflict & runtime centralization

Remote callbacks to vendor

Breaks the local trust model and privacy properties

Encrypted license logic

Prevents auditing, violates WRCode principles

Mandatory ledger verification

Unsafe in offline or degraded network environments

Anchors are always passive, read-only, and optional.

Legal Disclaimer

WRCode explicitly avoids the use of patented on-chain license enforcement techniques. All enforcement decisions are made locally, using pre-signed, auditable logic that resides within the WRStamped validator and never depends on external or executable on-chain code.

Summary

Anchoring in WRCode offers cryptographic transparency — not enforcement.

| Feature | Status |
|--------------------------|--|
| License status anchoring | <input checked="" type="checkbox"/> Optional |
| IOTA support | <input checked="" type="checkbox"/> Yes |
| Solana support | <input checked="" type="checkbox"/> Yes |
| Smart contracts used | <input checked="" type="checkbox"/> Never |
| Offline operable | <input checked="" type="checkbox"/> Always |

Anchors enhance trust. They never replace user control.

WRCode stands against the re-privatization of foundational cryptographic logic through misused software patents.

We fully recognize that cryptographic primitives such as hashing and digital signatures were groundbreaking inventions. But their core mechanisms have long since become part of the global security commons — openly specified, peer-reviewed, and essential to digital trust.

What we reject is the practice of repackaging trivial uses of these primitives — like basic hash comparisons — into vague, overbroad patent claims, often hidden behind abstract or technical-sounding language. This tactic creates artificial barriers, stifles open innovation, and enables legal pressure rather than real progress.

WRCode does not merely sidestep such claims — we actively resist them.

We document our methods publicly, use timestamped disclosures (e.g. OpenTimestamps), and publish defensive prior art where needed. This isn't just for transparency — it's a strategic defense against the weaponization of obvious logic.

Security must be verifiable, interoperable, and legally unencumbered.

It is not a privilege to be licensed — it is a shared responsibility to be defended.

Security is a right, not a business model.

🚫 No Obfuscation – WRCode Enforces Open Code Integrity

WRCode does not allow encrypted, obfuscated, or dynamically generated logic. To be eligible for WRStamping, all modules must be:

- Plain-text or readable PHP at the time of stamping
- Fully inspectable by WRGuard during pre-deployment analysis
- Immutable after stamping – no runtime code injection or self-modifying behavior

This is not about forced transparency of business logic, but about ensuring runtime verifiability and user safety.

📦 What Plugin & Theme Publishers Must Follow

-  Closed-source or obfuscated plugins are not eligible for WRStamping
 -  All child themes must be WRStamped separately, even if based on a WRStamped parent
 -  Code must be static, reviewable, and packaged in a trusted format
-

Why This Matters

WRStamp is a runtime trust contract. If users can't inspect or verify what will run, they can't trust it — even if it came from a known publisher.

This approach preserves user security, blocks malware injection, and prevents "hidden license logic" or unauthorized code execution.

Tools for Licensing Without Risk

To support commercial vendors, WRCode provides everything needed to migrate securely:

| Tool | Purpose |
|------|---------|
|------|---------|

| | |
|---|--|
|  | Plugin Signing Kit Securely package licensed logic into pre-approved modules |
|---|--|

| | |
|---|---|
|  | WRStamp CLI Stamp all files and anchor them to your vendor identity |
|---|---|

| | |
|---|---|
|  | Policy Loader Bind features, trials, or usage scopes to verified tokens |
|---|---|

Final Word

Licensing, feature control, and commercial protections don't need eval(). They don't need obfuscation. They just need integrity — enforced cryptographically and verified before runtime.

WRCode isn't limiting what you can offer. It's limiting how you offer it — in favor of security, trust, and long-term accountability.

 WRCode will make it easy to comply — but we will not lower the bar for compatibility.

 Lockdown Response If any WRStamp fails:

-  Credentials stay encrypted
-  DB access is denied
-  Plugins/themes are blocked
-  Logs and alert triggers fire
-  Execution is suspended unless integrity assertions are satisfied.

This is not a "warning plugin." This is execution enforcement.

 Why We Start With WordPress

Because WordPress powers over 40% of the internet — and remains one of the most targeted ecosystems for automated attacks and backdoor injections.

By starting with WordPress, WRCode:

- Protects the highest-risk surface
- Establishes the WRStamp standard where it matters most
- Creates an example that other ecosystems will follow

After WordPress, we will expand into:

- Laravel / Symfony (PHP)
- Node.js stacks (Express, Next.js)
- Python backends (FastAPI, Django)
- Other CMS

Summary: WRCode doesn't scan for threats.

It prevents them — by refusing to run untrusted code.

| Situation | Result |
|-----------------------------------|---|
| File is WRStamped | <input checked="" type="checkbox"/> Executes |
| File is missing/modified | <input checked="" type="checkbox"/> Execution denied |
| Plugin generates code at runtime | <input checked="" type="checkbox"/> Blocked unless pre-WRStamped |
| Plugin uses eval() logic | <input checked="" type="checkbox"/> Incompatible (use WRCode API/SDK) |
| Plugin uses signed static modules | <input checked="" type="checkbox"/> Fully compliant |
| Query called from trusted stack | <input checked="" type="checkbox"/> Executes |
| Query called from unknown origin | <input checked="" type="checkbox"/> Blocked |

⚖️ Plugin vendors can still protect their code, enforce licensing, and ship commercial features — They just have to do it transparently, verifiably, and securely.

🛡️ *WRCode doesn't eliminate all attack vectors — but it raises the bar so high that, when all guidelines are enforced, most automated exploit chains and common hacker tactics become practically obsolete.*

The described WordPress integrity enforcement mechanisms (e.g., WRGuard) are provided solely as illustrative embodiments that support secure execution of WRStamped code. They do not form part of the claimed inventive step and are based on widely known security practices (e.g., hash-based validation).

WRGuard for WordPress- Technical Flow

Outline

1. A manifest-driven pre-execution validation mechanism that ensures deterministic code-path integrity through cryptographic WRStamps.
2. WRLogin Plugin – A hardened login mechanism that strictly uses WRCode-based authentication. Simple QR code scans are not sufficient; only WRCode-compliant tokens are accepted for secure access.
3. A pre-execution code authorization token cryptographically bound to both a user identity and an origin region, the token being validated against a signed execution manifest before allowing the code to run.
4. WRStamped plugins, themes, template kits and core files are shipped as integrity checked packages containing a PDF with WRCode. A scan provides infos about the publisher, origin, last update, and AI templates that analyse the entire code file by file in a multi ai agent

orchestration against known vulnerabilities, incompatibility risks of the wrstamped stack and other threats like gdpr. Its highly customizable as the user controls the process. While this is not a conventional vulnerability scanner or security auditor, the orchestrator can leverage AI templates to analyze WRStamped plugin code for known compatibility issues, deprecated functions, or sensitive patterns prior to installation. These tests may also run periodically at runtime based on policy. However, it does not employ general-purpose threat modeling or behavior-based malware detection. The orchestration is modular and user-controlled — not bundled with proprietary models or patented threat intelligence feeds.

5. Encrypted Credentials Vault – Database access credentials are encrypted and only unlocked on successful verification.
6. The WordPress bootstrap process begins only if the stack matches the current WRStamped manifest. This pre-execution gating is enforced through a lightweight integrity wrapper designed for tamper-evident, self-hosted environments. his gating happens entirely within the WordPress/PHP environment using mu-plugins. It does not interact with the operating system's bootloader or UEFI Secure Boot.
7. WRGuard Dev accounts: Developer Toolkit for Plugin WRStamping – Allow plugin authors to generate signed WRStamps for secure distribution.

WRStamped Plugin and Theme Distribution with Orchestrated AI Oversight

WRStamped plugins, themes, and template kits are distributed as sealed integrity packages that include a WRCode-verified PDF, conforming to strict WRCode.org packaging standards. Each PDF contains a tamper-evident manifest displaying key trust metadata, including:

- WRCode dev account: the verified publisher identity (WRID), verified via domain ownership, email authentication, and proof-of-origin validation,
- the registered origin (e.g. domain, DNS hash, or organizational anchor),
- the visible WRCode, scannable directly from the PDF,
- the license scope (e.g. commercial, personal, time-limited),
- the last update timestamp and the current maintainer identity,
- declared stack dependencies (e.g. PHP 8.3, MariaDB version, theme framework), and
- a list of included modules with their WRStamped file hashes.

Before installation, the visible WRCode can be scanned to retrieve all relevant metadata and initiate a pre-installation integrity and compatibility analysis. Because each WRStamped component is pre-registered and version-locked, its known characteristics and compatibility history are available to the orchestrator. Findings are stored contextually, allowing the orchestrator to identify previously detected issues or mismatches across the declared stack.

When scanned or validated, the WRCode also triggers a background AI validation process confined to the WR CODE orchestrator, which holds full oversight over the deployed stack. The orchestrator has visibility into the WordPress core, file system, database schema, container runtime, server

configuration files, and recent log events. A customizable multi-agent AI orchestration layer performs a simultaneous, stack-aware assessment, analyzing the incoming package against the current runtime environment. Agents can assess vulnerabilities, performance bottlenecks, GDPR risks, or compatibility issues specific to the user's stack profile. All results are displayed in a predefined diagnostics grid within WR CODE's multi-screen interface, offering live, modular insight before and after deployment. The system is highly flexible and can be customized using plain text instructions and custom context embeddings. Tests can also be scheduled to run at predefined intervals to ensure continuous validation of the runtime environment.

Overview

Traditional WordPress installations are highly vulnerable to file tampering, plugin abuse, and credential exposure. This article introduces a hardened WordPress deployment architecture using:

- WRStamped file integrity verification
- Encrypted database credentials
- Pre-bootstrap runtime check (before WordPress loads)
- Failsafe lockdown mechanism upon integrity failure

This setup ensures that WordPress will not boot and credentials remain encrypted unless all critical components are validated cryptographically. Implements using only open source and published protocols.

Manifest Updates & Cryptographic Verification

Any change to the WordPress environment — such as installing a new plugin, updating existing code, or modifying the core — requires the WRStamped manifest to be updated. Once a new manifest is generated, it must be cryptographically verified and timestamped through Solana or a comparable tamper-evident ledger.

This ensures that:

- Only authorized changes are accepted and registered
- A traceable proof of the software state exists on-chain
- Future integrity checks have a canonical source of truth

If the manifest is not updated following a legitimate change, the system will block boot and encrypted secrets will remain locked.

WRGuard Plugin Overview

At the heart of this architecture is the WRGuard plugin — a hardened mu-plugin that fulfills multiple critical security roles:

- Acts as the WRCode-hardened login gateway, enforcing strict authentication via WRCode tokens only; users can either scan a valid WRCode or use an established pairing token during returning visits to initiate login
- Handles integrity verification for all WRStamped components
- Controls access to index.php, halting boot if verification fails
- Manages decryption of encrypted credentials, only unlocking secrets if system state is trusted
- Coordinates installation of new WRStamped plugins and updates to the WRStamped manifest

WRGuard is the central control mechanism that turns WordPress into a verifiable, tamper-aware platform. It executes early during boot and has full authority to allow or deny system operation based on verifiable cryptographic state.

 Architecture Overview

Request



index.php ←— Validates WRStamped Manifest before loading WordPress



 |
 |—— WRGuard::check_integrity()

 |—— WRGuard::decrypt_credentials()



wp-config.php ←— Receives DB credentials only if verified



WordPress Core + Plugins

 Component Breakdown

1.  WRStamped Manifest

To prevent tampering, the manifest itself must be protected against unauthorized modification. Simply storing it as a plaintext JSON file is not sufficient, as an attacker could forge hashes and deceive the system.

To ensure integrity:

- The manifest is signed using a trusted cryptographic key (e.g., Ed25519)
- The signature is verified before trust is established
- Both the manifest and credentials are encrypted during the initial installation phase to ensure full auditability and prevent post-deployment tampering. The manifest is only decrypted by authorized processes at runtime

To maintain auditability and security simultaneously, the WRGuard plugin provides a CLI/API function to finalize the current manifest, sign it, and submit it to a cryptographic ledger. This creates a tamper-evident, traceable fingerprint for future validation.

No updates to the manifest are accepted unless this controlled sealing process is explicitly triggered.

This system anchors stack manifests on Solana for public auditability. WRGuard does not use smart contracts or runtime feature gating, making it more lightweight than blockchain licensing systems. Each release of your WordPress distribution is signed with a WRStamped manifest, containing cryptographic hashes of:

- Core WordPress files (e.g. wp-includes/, wp-admin/)
- WRStamped plugin, theme or template kit distributions, which are verifiably signed by their respective plugin providers

Example manifest:

```
{  
  "version": "6.5.3",  
  "hashes": {  
    "index.php": "a7f9...",  
    "wp-includes/functions.php": "44bc...",  
    "wp-content/plugins/wrlogin/main.php": "fae3..."  
  },  
  "signature": "ots://abc123"  
}
```

The manifest is either:

- Embedded in the bundle
- Pulled from a trusted registry
- Timestamping is executed through batch-anchoring on the Solana blockchain, enabling verifiable immutability.

2. 📁 index.php Guard Layer

Your index.php is wrapped to block execution unless all checks pass:

```
require_once __DIR__ . '/wr-guard/bootstrap.php';
```

```
if (!WRGuard::check_integrity()) {  
    http_response_code(503);  
    exit('System integrity check failed. Execution halted.');  
}  
  
require __DIR__ . '/wp-blog-header.php';
```

 This ensures that no part of WordPress loads unless the integrity check passes.

3. Encrypted DB Credentials (.wrvault/db.enc)

The database credentials are not stored in plaintext inside wp-config.php.

Instead, they're encrypted with Libsodium (crypto_secretbox) and stored in a separate file:

Encrypted file:

.wrvault/db.enc

Decryption only occurs if:

- All files pass hash verification
- Optional: A local hardware key or WRCode is detected
- The decryption key is accessible in the runtime environment (e.g. TPM, secure env var)

4. wp-config.php Partial

Only calls the decryption API if WRGuard passes:

```
require_once __DIR__ . '/wr-guard/bootstrap.php';
```

```
if (!WRGuard::check_integrity()) {  
    die('System integrity check failed.');//  
}  
  
$creds = WRGuard::load_db_credentials(); // returns associative array  
  
define('DB_NAME', $creds['name']);  
  
define('DB_USER', $creds['user']);  
  
define('DB_PASSWORD', $creds['pass']);  
  
define('DB_HOST', $creds['host']);
```

This means: no file or plugin alone can retrieve credentials, even if compromised.

 Implementation Notes Credential Encryption via Sodium

Encryption:

```
$nonce = random_bytes(SODIUM_CRYPTO_SECRETBOX_NONCEBYTES);

$ciphertext = sodium_crypto_secretbox($jsonData, $nonce, $key);

file_put_contents('.wrvault/db.enc', $nonce . $ciphertext);
```

Decryption:

```
$data = file_get_contents('.wrvault/db.enc');

$nonce = substr($data, 0, SODIUM_CRYPTO_SECRETBOX_NONCEBYTES);

$ciphertext = substr($data, SODIUM_CRYPTO_SECRETBOX_NONCEBYTES);

$plaintext = sodium_crypto_secretbox_open($ciphertext, $nonce, $key);

return json_decode($plaintext, true);
```

 Hash Verification Logic

```
function check_integrity(): bool {  
    $manifest = json_decode(file_get_contents(__DIR__ . '/../wr.manifest.json'), true);  
  
    foreach ($manifest['hashes'] as $file => $expectedHash) {  
  
        $path = realpath(__DIR__ . '/../' . $file);  
  
        if (!file_exists($path) || hash_file('sha256', $path) !== $expectedHash) {  
  
            return false;  
        }  
    }  
  
    return true;  
}
```

 What Happens When a Hash Fails?

If *any* file is altered:

- Credential access is governed by a WRStamped policy that checks stack integrity prior to decryption. This mirrors common zero-trust practices, adapted for user-space orchestration in WordPress
- index.php immediately stops execution with an HTTP 503
- Optional: Admin gets notified via webhook or local log

👁 Enhancements

| Feature | Description |
|-------------------------------|---|
| YubiKey / WRCode login unlock | Unlock decryption key only after WR scan |
| Cryptographic logging | Public tamper-evident logging of releases |
| Hardware-bound key | Key never stored, only derived on secured device |
| Multi-plugin WRVerification | Allow plugin authors to register their own WRStamps via CLI |
| mu-plugin lockdown | Only enable WRLogin plugin if system is verified |

 Why This Matters

Most WordPress infections work by:

- Replacing or injecting into functions.php, index.php, wp-includes/
- Reading plaintext DB credentials from wp-config.php
- Installing fake or malicious plugins

This architecture blocks all three vectors by:

- Detecting unauthorized code changes
- Refusing to decrypt secrets
- Stopping execution before WordPress even boots



🛡 WRGuard: Technical Overview of Tamper-Proof Plugin & Theme Integrity in WordPress

Legal Note: WRGuard integrates well-established security principles—including file hashing, credential gating, and rollback enforcement—into a modular orchestration framework tailored for WordPress. No claims of novelty are made for these individual mechanisms. Instead, WRGuard’s strength lies in the composable, verifiable, and orchestrated deployment of these techniques within a trust model. WRGuard is a policy enforcement layer; it does not claim novelty in integrity enforcement mechanisms. The system is transparent and modular by design. WRCode is a voluntary trust protocol. While anyone may use, copy, or modify WRCode templates freely, the official WRCode infrastructure — including WRGuard and orchestrator features — will only process templates that are WRStamped and linked to a verified WRCode account. This protects users by ensuring origin traceability, version control, and tamper-proof distribution. Forks of WRCode may choose to adopt or ignore these policies, as the code is open and extensible. WRCode doesn’t limit software freedom. It limits access to cryptographic trust infrastructure. Anyone can run the code — but only licensed WRCode accounts receive the benefits of tamper-proof orchestration and runtime verification. WRCode redefines automation licensing. Instead of encrypting code or enforcing proprietary runtime checks, WRCode ties execution to verifiable templates and user-controlled trust contexts. Each template carries its own proof of origin, usage scope, and integrity — optionally anchored on a

public ledger. Execution is allowed only if the runtime stack, user vault, and WRStamped policy match. It's a shift from code ownership to orchestration participation.

WRGuard is a deterministic verification layer that enforces pre-execution code integrity across the WordPress stack — including themes, plugins, and even the WordPress core. It is part of the WRStamped ecosystem and uses a Merkle Tree–inspired structure for scalable and deterministic verification. WRGuard is not a security innovation in isolation, but a mechanism to enforce WRCode-anchored integrity in user-controlled WordPress environments.

- Full Code Traversal

When a plugin, theme, or update is installed or modified, WRGuard:

- Runs a recursive foreach loop through every file.
- Breaks each file into code blocks.
- Hashes each code block individually.

- Merkle Tree Architecture

- Each hashed code block is linked to its parent file (root).
- Files are grouped under their plugin/theme/app or WP core (top root).

- The root hash of this Merkle-style hashing tree (root hash) is generated as a deterministic digest of the code stack.
- WRGuard applies a layered hashing scheme inspired by Merkle principles, optimized for modular integrity workflows within the WRCode model. These hashes are anchored via open, timestamped registries (e.g., Solana) to support trustless validation.
- Only the original publisher can WRStamp a plugin, theme, or app.
- Stamping requires verified identity (including origin country) and cryptographic anchoring.
- Hashes are immutable and traceable.

 WRGuard: Distributed integrity verification (WRCode account bound)

 Outline

1. Distributed Hash Verification Process
2. WRStamped Core Hashing
3. Credential Lockdown & Decryption Protocol
4. Runtime Stack Guard with Fail-Safe Mechanism

5. Developer-Focused Recovery Mode & Patch Pipeline

Enhanced Verification: Distributed Hashing Layer

Unlike traditional models that rely solely on file checks during runtime, WRGuard now utilizes a dedicated verification process that mirrors the pre-deployment hashing logic:

- The same function that hashes the stack pre-deployment is executed post-deployment during runtime.
- Hashing can occur locally or in a distributed cluster (LAN or orchestrated external nodes).
- Each integrity check may take ~2s, but multiple staggered intervals (via external nodes) allow continuous rolling checks without blocking user interaction.
- Results are verified against the cryptographic-anchored WRStamped baseline.

Stack Lockdown on Mismatch

When a hash mismatch is detected:

- Affected code can optionally be immediately encrypted and set to read-only mode, especially in high-security environments. This ensures that tampered components cannot be altered, reloaded, or reverse-engineered without explicit developer access via WRGuard Dev Mode.

Dev Recovery & WRGuard Developer Mode

If tampering is detected, WRGuard Dev Edition activates:

- A dynamic difference report is generated, highlighting:
 - What changed (file/code block level)
 - Time of change
 - Suspected origin (if detectable)
 - Linked WRCode identity (if change attempt used token)

Automated Git-Based Recovery & Repair

With the WRGuard Dev Tools:

- Developers can unlock vaults using their WRCode.

- Git credentials are securely injected into a session.
- A CLI-based command allows pushing a WRStamped fix commit to the repository.
- Optionally, patches can be reviewed and enhanced using AI agents (e.g., via Cursor or the orchestrator's optimization layer).

The orchestrator performs:

- Code context analysis
- Automated fix validation
- Runtime test stubs (where supported)
- Dynamic WRCode generation for deepfix mode

Merkle Tree-Based Verification Principle (Core Hashing)

At the heart of WRGuard's architecture is a Merkle Tree-inspired hashing model:

- Every file in the stack (plugin, theme, core) is recursively scanned.
- Each code block within a file is hashed individually.
- These hashes are grouped and linked to the parent file ("root").

- The parent file hashes are then linked to a higher-level root hash ("top roof" or stack root).
- The final root hash is WRStamped and anchored to the ledger, creating a tamper-evident and immutable reference.

 This structure ensures:

- Every single block of code is registered before execution.
- Only issuers with a valid WRStamp are permitted to register new code blocks.
- No unregistered or foreign code can run, even if injected at runtime.
- Subsequent fixes can be applied at the block level, allowing more precise recovery without replacing entire files.
- While top-level root hash checking provides a lightweight checkpoint for quick validation, WRGuard operates with a dedicated verification process that replicates the full pre-deployment hash logic in a separate background thread or distributed process. This approach ensures that:
 - All roof hashes (per file) and code block hashes are re-calculated continuously.
 - Unauthorized changes are pinpointed precisely at block level.

- Detected differences are automatically analyzed and mapped into a template map dynamic WRCode for WRGuard Dev Edition.
- This template map enables generation of tailored recovery templates, which the developer can adapt, overwrite, or register as a personal template.

If a personalized fix is used and intended to re-enter the public WRCode ecosystem, it must be re-WRStamped by the original issuer or verified via an alternative secure flow. Private solutions remain confined to the optimization layer and are not included in public WRCode validation logic.

To maintain high-frequency assurance, distributed integrity checks can run every 2 seconds or even faster, depending on the number of verification nodes available. These parallel hash evaluations ensure continuous stack surveillance without central bottlenecks, increasing trust and resilience in real time.

- WRGuard Core Plugin blocks execution at:
 - index.php
 - wp-config.php (credential access)
 - .htaccss

- For high-assurance deployments, WRGuard supports an optional lockdown mode that halts runtime memory execution upon tamper detection. This approach is configurable and intended for regulated or mission-critical environments. This ensures no malicious payload can remain active in memory after code compromise.
- Dev tools (restricted access)

🛡 Updated Component Workflow

1. WRGuard Hash Agent (local/clustered) performs:

- Full stack hash traversal
- Cross-check with WRStamped manifest (ledger-based)

2. index.php Guard halts WP load:

```
require_once __DIR__ . '/wr-guard/bootstrap.php';

if (!WRGuard::verify_remote_hash()) {

    http_response_code(503);

    exit('Integrity failed. Stack sealed.');

}
```

3. wp-config.php Decryption Lock:

```
if (!WRGuard::verify_remote_hash()) {  
    die('Integrity check failed.');//  
}  
  
$creds = WRGuard::load_db_credentials();  
  
define('DB_NAME', $creds['name']);
```

4. WRGuard Dev Console Activation:

- Accessible only from CLI or localhost
 - Provides a detailed templatemap + difference view
 - Generates a WRCode dynamically that includes all required AI instructions for analysis and bug-fixing workflows. Secure Git access credentials to be stored in the WRVault, enabling automatic push of the synthesized code fix after orchestration and validation by the AI optimization layer.
-

Why It Matters More Than Ever

WRGuard now:

- Eliminates runtime drift & plugin-level exploits.
- Enables real-time validation without performance hit.
- Secures credential access via binding decryption to verified state.
- Enables rapid patching via WRCode + Git-based DevFlow.
- Forms the foundation for scalable plugin & app WRStamp ecosystems.

Runtime Integrity Verification via Containerized Inspection

To ensure a tamper-proof and audit-ready runtime environment, WRGuard performs containerized stack verification at fixed intervals, using hash-matching logic to ensure integrity. This container includes a PostgreSQL instance that supports fast hash comparisons, secure decryption of critical logic, and immediate triggering of integrity events.

The verification process covers the entire `htdocs/` directory (referred to as the rooftop layer) and recursively hashes every file and code block, excluding only the `uploads/` directory to avoid false positives from user-generated content. Each element—down to the smallest code segment—is

hashed and compared against its WRStamped reference. Any unauthorized modification or insertion breaks the hash chain and flags the system for review.

Enhanced Integrity Monitoring and Real-Time Automation Control

Upon detection of a mismatch, the system generates a detailed JSON diff report that includes:

- Templatemap outlining the affected automation components
- Delta snapshot highlighting file- or logic-level changes
- WRCode embedding dynamic context metadata (e.g., timestamp, origin, change scope)
- Developer-bound token, which restricts visibility and restoration rights to verified developer accounts (WRGuard Dev Accounts)

Additionally, a Dockerized integrity checker—typically consisting of a PostgreSQL instance and shell script—is responsible for scanning not only the file system but also the WordPress MySQL database for signs of critical injections (e.g., SQL-based payloads or unauthorized admin entries).

When connected to the orchestrator with the optimization layer enabled, this setup allows for permanent live-time analysis of both code and database state.

- If a critical injection is detected, the system triggers an immediate shutdown.

- If questionable activity is detected, the corresponding events can be transmitted to the orchestrator—assuming the WRGuard Dev plugin is active on the site, the user is authenticated, and the orchestrator is actively running in the browser session.

A lightweight local LLM (e.g., Mistral 7B) may be utilized in the background to perform context-aware anomaly detection and behavior classification.

To maintain trust in static components:

- Static database tables can be individually hashed immediately after installation or update to establish a trusted integrity reference.
- Hashes are stored and cross-referenced against WRStamped verification records.
- The orchestrator itself participates in the validity chain but cannot fully assert integrity when executed in a browser environment with non-verified plugins or extensions. In such cases:
 - Users may WRStamp (User Stamp) trusted versions.
 - Providers can be whitelisted to push verified updates.
 - Upon each update, the system prehashes the package and performs mathematical validation against the publisher's WRStamped keychain.

Looking ahead, once WRCode adoption increases, we aim to encourage browser and extension publishers to adopt the WRStamped standard—enabling a fully verifiable integrity chain across the full execution environment.

If WR CODE orchestration is active, the JSON can be immediately pushed into the orchestrator as it will be directly compatible. This allows AI agents to react, document, or remediate in real time—depending on the configured policies and automation permissions.

This architecture ensures that no plugin, theme, or injected logic can operate outside the verified trust boundary—while keeping the system fast, modular, and privacy-aware.

Advanced users can extend the integrity checks to include critical server-side files such as configuration files, shell scripts, or container manifests. In parallel, server logs—such as access logs, authentication events, or system anomalies—can be read in real time and pushed directly into the WR CODE orchestrator alongside the integrity report, enabling deeper analysis, automated triage, and AI-assisted remediation.

This allows the orchestrator to correlate anomalies across different layers—codebase, infrastructure, and behavior—transforming local integrity breaches or suspicious log patterns into actionable insights and threat intelligence across the entire software stack.

No WordPress system should boot unless all hashes are accounted for, signatures are trusted, and changes are recorded.

With WRGuard, you're not just running code — you're verifying trust at the deepest possible level. In a world where cyberattacks now cost organizations over \$10 trillion globally per year and where breaches cripple critical infrastructure, businesses, and even essential services, the status quo is no longer acceptable. The ripple effects go far beyond IT: from fuel supply disruptions and water treatment failures to economic and ecological harm, the cost of insecure systems is staggering.

WRGuard enables a future where software is not only open but also auditable, traceable, and tamper-proof by design. No hidden logic. No unverified sources. Only trustable code, cryptographically bound to verified identities — and running under your control.

Though still early in its evolution, WRGuard is being actively developed to establish a new standard of verifiable software trust. The need has never been more urgent.

Runtime Token Verification and Database Enforcement Layer

In the WRGuard architecture, every database-affecting action (insert, update, delete) proceeds normally through the standard WordPress execution flow to ensure that frontend performance

remains unaffected. These operations are not blocked or delayed, but they are accompanied by a WR_TOKEN—an ephemeral token regenerated every 5 minutes and derived from the top-level Merkle root (root token) of the active WRStamped code stack.

To correlate stateful database operations with verified runtime components, ephemeral tokens derived from the current WRStamped state are attached to metadata fields. These enhance traceability without introducing protocol-level enforcement. To enable this, the respective MySQL tables are automatically extended during WRGuard installation to include token-tracking fields (e.g., wr_token, wr_origin), making them ready for token-aware enforcement without requiring structural changes by plugin or theme developers.

In parallel, a PostgreSQL-based integrity service asynchronously mirrors the affected MariaDB tables and continuously monitors incoming changes.

If a write is detected without a valid WRStamp, the system logs the event and triggers a customizable audit workflow. This includes automatic diff generation, token validation, and rollback preparation. Unauthorized writes are later remediated by a post-commit script that restores verified state from a PostgreSQL-mirrored copy. This script, configured per deployment, functions as a real-time "firefighter"—restoring the last verified state or quarantining modified records within milliseconds of detection. WRGuard uses proven database journaling techniques with WRToken

correlation to track and revert unauthorized writes. This system integrates trust anchors rather than proposing a new database standard.

The PostgreSQL enforcement layer retains full snapshots and field-level diffs of all unauthorized operations. These are relayed to the orchestrator for further analysis and optionally linked to session data or WRGuard developer accounts.

The practical outcome is that no unstamped code can cause significant damage: untraceable or cascading follow-up actions are stopped immediately at the root. This enables instant diff-based diagnostics and deepfix routines—while keeping the system highly responsive and fully compatible with existing plugins and themes, without requiring any code rewrites.

Note: This feature uses standard open source database logging and audit capabilities (e.g., PostgreSQL journaling and triggers). No proprietary rollback algorithms or patented database logic is introduced.

Pre-Execution Blocking Mode for High-Assurance Environments

WRGuard's default trust model is designed to deliver maximum security with zero friction. In this mode, all database-affecting operations (insert, update, delete) proceed normally through the standard WordPress execution flow, ensuring full compatibility with themes, plugins, and

performance optimizations. If a write is found to be unauthorized—i.e., it lacks a valid, time-bound WR_TOKEN derived from the WRStamped code stack—it is automatically reverted within milliseconds, and the event is logged and audited.

This approach ensures that even if an unauthorized write occurs, it cannot persist long enough to cause downstream effects. In practice, the write is reversed before any external system, user process, or chained automation can react. For the vast majority of WordPress use cases—including e-commerce, content platforms, and SaaS frontends—this reactive model already offers maximum practical protection without impacting the user experience or breaking compatibility with legacy components.

However, in high-assurance environments, where even transient writes must be ruled out completely—such as financial systems, legal records, or critical infrastructure—WRGuard supports an optional pre-execution blocking mode. In this stricter configuration, every database-modifying query is intercepted before it executes, and only allowed to proceed if it includes a valid WR_TOKEN verified against the active Merkle-root state.

This ensures that no unauthorized action ever reaches the database, offering absolute runtime integrity. Blocking occurs at the query interface level using database-native mechanisms or hardened proxies. Here, PostgreSQL is especially well-suited, as it provides robust support for custom triggers, inline validation, and enforcement layers that can run side-by-side with the orchestrator's audit

services. Critically, this strict mode does not need to be applied globally. WRGuard allows selective enforcement on a per-table basis, enabling strict blocking for sensitive data (e.g., `wp_users`, `wp_usermeta`, `wp_options`) while using the fast rollback model for general content (e.g., `posts`, `meta`, `logs`). This hybrid approach delivers maximum protection where it matters, without sacrificing compatibility or ease of adoption elsewhere. The strict mode is not enabled by default, and for good reason: it adds complexity, may require compatibility adaptations, and can introduce delays or errors in legacy systems that were never built with such enforcement in mind. WRGuard's default mode already ensures that no unauthorized action can cause harm, thanks to sub-second rollback, audit logging, and orchestrator-level remediation. In other words: security is enforced by default—strict mode is available when policy demands it.

This level of flexibility is critical for encouraging adoption. By allowing site owners, developers, and enterprises to progressively harden their systems—from rollback enforcement to selective blocking to full zero-trust enforcement—WRGuard helps pave the way for a next-generation secure internet. One that's built on verifiable actions, cryptographic integrity, and trustless automation—without compromising usability.

Note: This feature uses standard open source database logging and audit capabilities (e.g., PostgreSQL journaling and triggers). No proprietary rollback algorithms or patented database logic is introduced.



Code



Email



Contact forms

„Security is only as strong as its weakest link”

WRGuard secures email, contact forms, and
code through a unified trust model

Why Verified Attachments and Contact Forms Matter: WRGuard as a New Standard for Secure Communication

Page 1: The Risk Behind "Normal" Email and Contact Form Submissions

For years, businesses have operated under the illusion that as long as attachments are not clicked, or emails and form submissions come from seemingly legitimate sources, there's little danger. This is a dangerous myth.

Every year, billions of dollars are lost to phishing, PDF-based malware, impersonation attacks, and document tampering — many originating from unsecured contact forms or fake file attachments. While firewalls and antivirus systems help, they act only after the fact. WRGuard takes a fundamentally different approach: it rejects unauthenticated communication before it enters your system.

The Problem

- Attachments are unauditible. Anyone can send a PDF pretending to be a quote, invoice, or legal document.
- Contact forms are spoofable. Most sites accept unauthenticated, user-controlled input and attachments.

- Sender identity is easily forged. Even with SPF/DKIM/DMARC, attackers can fake names, logos, and formatting.
- Content can be obfuscated. PDFs and even text fields can contain hidden scripts, Unicode trickery, or misleading formatting.
- Automation is risky. Systems that auto-process submissions (OCR, parsing, AI triggers) are exposed to untrusted data.

WRGuard's Approach: Trust by Default

WRGuard flips the model:

- Only WRStamped attachments and WRGuard-verified contact form submissions are accepted.
- Each WRStamped payload includes a cryptographically signed trust token with timestamp, origin data, and template definition.
- Contact form submissions are composed in the Secure Layer or via a verified WRComposer.
- Any unstamped input — whether form or file — is rejected *by design*.

This prevents:

- PDF tampering

- Spoofed form submissions
- Hidden scripts or stealth payloads
- Replay or impersonation attacks

If it's not verified, it doesn't exist.

WRGuard unifies previously disconnected domains — file uploads, form submissions, license-based customer routing, and automation triggers — into a single, composable trust model. This architecture is extensible to CRMs, ticketing systems, and secure workflow orchestration.

How It Works + What Admins Should Do

WRGuard-secured contact forms can also include an optional field for AI instructions. These instructions are only accepted if submitted within a WRStamped context and are strictly limited to automation flows predefined by the website owner. The available AI actions are transparently defined in the verified template, and these templates can differ dynamically depending on the user's relationship, license level, or previous context — all verifiable via cryptographic anchoring. Users see a personalized version of the form, including what automation actions are available to them, based on their verified customer status — such as summarizing input, routing requests, extracting structured data, booking appointments, analyzing queries, or triggering follow-up messages.

In advanced cases, the AI logic can even analyze uploaded WRStamped PDFs or related context during submission, providing immediate feedback if critical information is missing or incomplete. This enables proactive guidance for the user, ensuring submissions meet internal requirements before being routed to the appropriate department. The result is a secure, intention-driven user experience, with automation possibilities tailored and made visible up front. And because both the sender and receiver systems can operate with the WR CODE optimization layer in parallel, this enables proactive workflows to run in real time — validating input, pre-processing data, and routing it intelligently as it's received. In time-critical environments, every minute counts — and WRGuard ensures that no trusted opportunity is delayed or lost due to manual intake bottlenecks.

Before a WRGuard-secured form is accessed, the system verifies user authenticity through one of two trusted entry methods:

WRCode Scan or Secure Entry: Users can access protected forms by either scanning a dynamic WRCode displayed on the site or arriving from a device already paired via the WRLogin system.

Session-Aware Access: If the user is already authenticated within the orchestrator's Secure Layer, the form unlocks automatically. Returning users with token-bound devices are auto-logged in or auto-registered based on their configured preferences. WRGuard manages this seamlessly through its built-in WRLogin integration, ensuring secure and frictionless access.

This ensures the form only becomes interactive within a cryptographically trusted context, allowing preloaded profile data, license verification, and secure AI interactions to be applied immediately.

1. Secure Layer Submission

- Users fill out WRGuard-secured contact forms or use WRComposer.
- Attachments and text fields are composed in a controlled, isolated environment.
- The form only functions once the user is connected through their registered WRVault. At that point, the license level — if WRStamped and anchored — can be made available for verification with the user's consent. While license tier data is generally non-personal and serves to enable access-level validation or routing logic, WRGuard still requires user approval before exposing this information — ensuring full GDPR compliance and maintaining maximum transparency. All other attributes — such as identity, preferences, PII, contextual memory, or behavioral signals — remain under user control and are only shared with explicit consent. Nothing is transmitted without prior consent. This ensures that all data exchange is privacy-by-design, transparent, and fully under user control.

2. WRStamp Applied

- A cryptographic hash of the content + template + metadata is generated.
- The WRStamp includes timestamp, template ID, and origin fingerprint.

- Licensing or user-tier data can be verified *during scan* — enabling dynamic policy enforcement or tailored automation (e.g., prioritizing support for enterprise clients).

3. Anchor and Transmit

- The WRStamp is anchored (e.g., on Solana).
- The payload is transmitted or stored — the recipient verifies the WRStamp before acting.

Example WRStamp payload:

```
{  
  "template_id": "contact-basic-v1",  
  "hash": "cafe123...",  
  "origin": "vault:device123",  
  "timestamp": "2025-07-20T10:00:00Z",  
  "anchor": "solana://...",  
  "license": "verified-enterprise"  
}
```

Admin Guidelines

- Enable WRGuard Contact Form Protection in the WRGuard plugin.
 - When enabled, your forms will only accept submissions composed in the Secure Layer or via a valid WRCode pairing.
 - Submissions without WRStamps are blocked, and optionally logged or quarantined.
- Never open attachments that are not WRStamped.
- Treat all unverified contact messages as untrusted by default.
- Use the WRGuard plugin to verify stamps automatically.
- Whitelist only WRComposer-verified input.
- Leverage license-bound routing rules to provide differentiated response flows.
- Log and quarantine anything that fails verification.

Recommended Auto-Responder

Subject: Submission Rejected – Verification Required

Your message was received but could not be verified.

For your protection and ours, we only accept WRGuard-verified forms and documents.

Please resend your inquiry or file using our secure WRGuard form: [yourdomain.com/contact-wrsecure]

This process ensures sender authenticity, cryptographic integrity, and auditability.

Thank you for supporting secure and tamper-proof communication.

Final Note: Security is only as strong as its weakest link — and that link is often overlooked.

Whether it's an email attachment, a contact form, or embedded code from a third-party plugin — each element in the communication chain must be verifiable, auditable, and resistant to manipulation. WRGuard ties these layers together under a unified trust architecture, ensuring that every interaction — from incoming documents to submitted data — is cryptographically validated and contextually controlled.

This isn't simply an upgrade. It's a structural shift toward zero-trust, policy-bound automation where identity, content integrity, and permission models are all cryptographically enforced. WRGuard gives businesses the tools to operate with confidence, build automation on solid ground, and prove — not assume — trust.



WRGuard Integrity Backup Module – MinIO WORM Integration

To bring enterprise-grade integrity protection to the open source WordPress ecosystem, WRGuard includes native integration with MinIO WORM (Write Once Read Many) storage into the concept. This system enables tamper-proof archiving of verified file and database states, complete with cryptographic anchoring and orchestrator reporting.

File System Backup Flow

- On manual or scheduled backup, WRGuard performs a complete integrity check on all WordPress-related files.
- If the entire file tree matches the expected WRStamped hash map:
 - A verified snapshot is cloned into MinIO WORM storage, ensuring it cannot be modified afterward.
 - The corresponding snapshot hash is anchored on IOTA, providing long-term public verifiability.

 Database Backup Flow

MariaDB:

- A WRDiff analysis checks for unauthorized data entries (e.g., via plugin exploits or direct injection).
- If no unauthorized data is detected:
 - A clean snapshot is taken.
 - InnoDB's built-in integrity checks verify internal consistency and detect corruption from hardware faults.
 - If passed, the snapshot is pushed into WORM storage and anchored on IOTA.

PostgreSQL:

- Integrity validation includes checksum and backup verification routines (`pg_checksums`, `pg_verifybackup`).
- Verified states are archived in MinIO WORM and anchored on IOTA, just like with MariaDB.

 MinIO Cluster Support & Redundancy

MinIO can be deployed in a distributed cluster configuration, enabling:

- Redundancy across disks and nodes
- Self-healing capabilities in case of hardware failure
- Horizontal scalability with minimal overhead

This ensures that even in the event of local storage failure, the backup remains recoverable without human intervention.

Optional MDISK Export

For enhanced offline resilience, users are encouraged to export confirmed WORM snapshots to MDISK or other immutable physical media. This provides long-term cold storage that remains usable even in catastrophic failure or regulatory investigations.

Orchestrator Reporting

Every verified backup generates a detailed report that includes:

- Hashes of all backed-up components
- Verification and integrity check logs
- IOTA anchoring receipts

- Timestamps and diff snapshots (if applicable)

This report is then pushed to the connected orchestrator, enabling distributed trust validation and full audit transparency across systems.

Open Source Deserves Real Security

Integrity validation, tamper-proof backups, and cryptographic anchoring are standard in regulated proprietary systems — but largely absent from most open source stacks.

WRGuard changes that.

By combining open verification with simple, automated tooling and resilient storage backends, it enables small teams and solo developers to achieve a security posture previously reserved for large enterprises.

Security shouldn't be a luxury — and open source shouldn't fall behind.

Visionary Concept: WRScan-to-Install

Beyond its core mission of verifiable automation, WRCode may one day extend into endpoint protection. The idea is simple: no new executable or browser add-on should ever install silently. Instead, the operating system would run in a locked-down state, where any unknown file is blocked

by default. A WRCode scan could serve as the unlocking step: the user scans a code tied to a signed manifest, verifying the publisher, file hashes, and intended purpose. Only then would the system permit installation. In practice, this could mean two layers of defense:

1. OS-level lockdown that prevents unapproved executables or extensions from running.
2. WRScan approval that allows exactly the intended package or add-on to proceed.

Possible technical approaches:

- Using Windows Defender Application Control (WDAC) to enforce code integrity across EXEs, DLLs, MSIs, scripts, and drivers, combined with strict enterprise policies to limit browser add-on installations.
- As an alternative or complementary measure, an automatic renaming mechanism could mark unknown files (e.g., changing .exe to .wrblock) so they cannot run until explicitly unlocked through a WRScan verification step.

The result would be a “scan-to-install” model, where silent malware installations simply never happen, and every unlock leaves an immutable audit trail. *Disclaimer: This is an exploratory concept under consideration. Its feasibility depends on technical constraints of operating systems and browser vendors, and is not part of WRCode’s current roadmap.*

Visionary Concept: Multi-Agent Policy Enforcement

Imagine security that works silently in the background. With WRGuard, a multi-AI agent workflow could act as a digital security team that never sleeps. Each agent focuses on a different task: one verifies file origins, another checks policy, a third monitors outbound traffic, while others sandbox new apps or record every decision in a tamper-proof log.

Focused scrutiny where it matters most

Not all software carries the same risk. Newly approved or freshly installed applications are the most likely attack vectors. The workflow could apply extra scrutiny during this critical period — running targeted anti-virus scans, watching outbound traffic more closely, and limiting network or file system access until the app proves trustworthy. Over time, as the software behaves consistently, the restrictions could relax.

Adaptive and environment-specific

Because the workflow runs in the background, the user never sees complex prompts or warnings. Unknown files, unexpected installs, or suspicious traffic are simply denied unless explicitly unlocked via a WRScan. At the same time, the workflow can be tailored to specific environments — a home user may allow consumer apps, while a factory workstation might only permit industrial control software and approved update servers.

If enabled, the system could also track local user behavior such as visiting websites or opening emails. These events would automatically trigger backend checkups, ensuring that attachments, downloads, and active sessions are immediately validated. This way, nothing ever has a chance to slip through unnoticed.

Simple for the user

For the end user, the experience stays effortless:

- Only WRScan-approved software installs.
- New apps are watched more closely.
- Web and email actions trigger automatic security checks.
- Everything else is silently blocked.

This approach turns endpoint protection into a quiet background safeguard, directing the most attention where it's most needed while keeping systems locked down and user experience simple.

Disclaimer: This is a forward-looking concept, not part of WRCode's current roadmap. Its realization depends on technical integration with operating systems and vendor ecosystems.

Future Work: Exfiltration Prevention & Forensic Traceability

WRCode is conceived as a modular framework. Its initial release focuses on pre-runtime verification and policy enforcement, while additional capabilities may follow as the project evolves. One such direction is exfiltration prevention combined with forensic traceability, aimed at high-risk or IP-sensitive environments. The principle is simple: WRStamped data should not leave a device unless transferred to an authorized WRCode account with the required permissions. All other channels—email, cloud services, removable media—would be blocked by default under a fail-closed model. When data is legitimately shared with another authorized WRCode account, it would be transmitted only in encrypted form, ensuring that no exfiltration or interception can occur in transit.

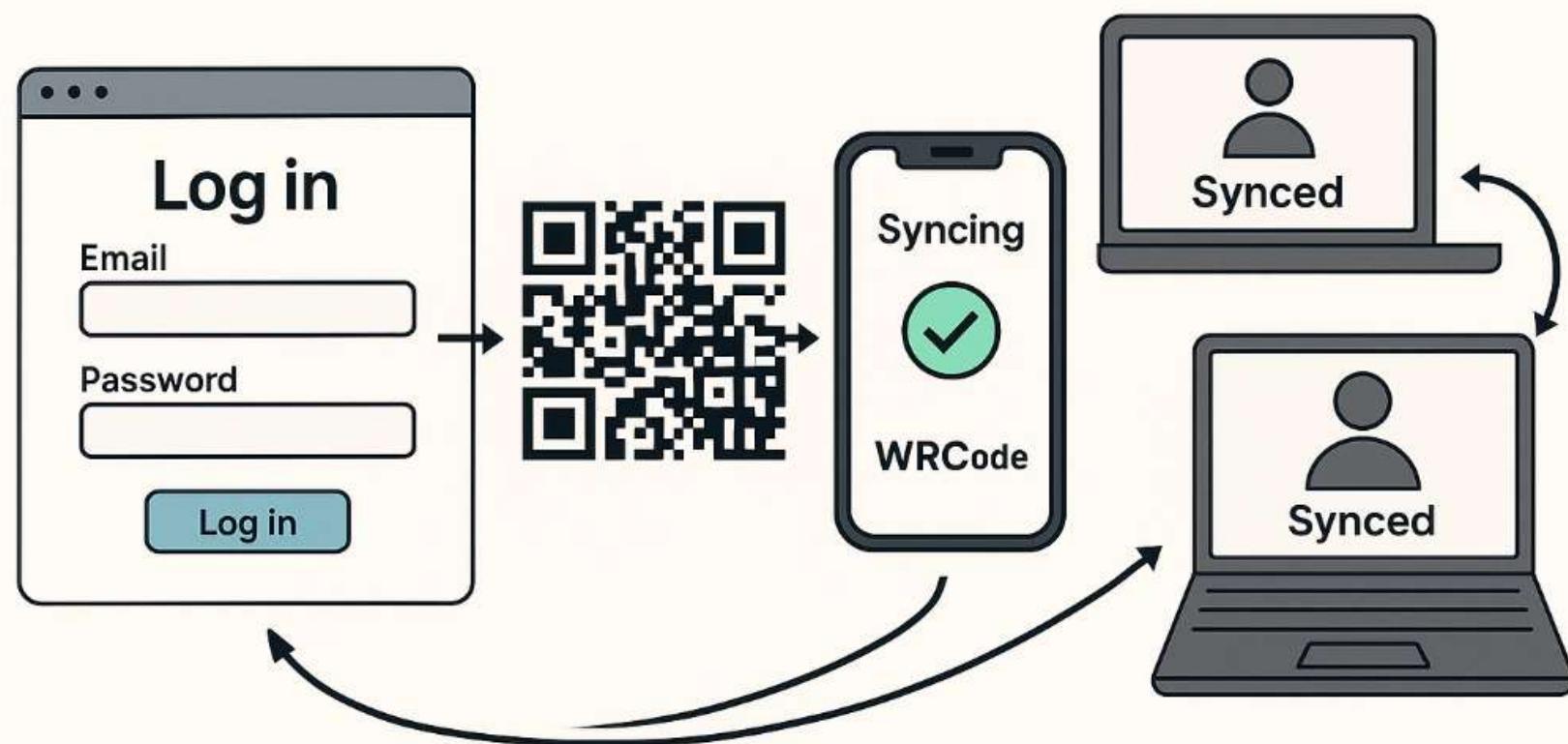
WRCode markers will be embedded directly into rendered documents, repeated in white font across the entire content. Invisible to the reader, these markers can be toggled on by the viewer, making them scannable. Each marker carries a payload, such as session ID and account owner, but could also embed AI automation instructions, additional context, or memory data (e.g. who viewed or modified the document). Even if a fragment is leaked through screenshots or photography, the marker would remain detectable and attributable. In this way, the text itself can carry a double meaning: while readable in plain language, it simultaneously embeds scannable WRCode patterns. The dedicated WRCode viewer software would be able to decipher these hidden codes and extract the embedded WRCode payloads, ensuring that deeper instructions, metadata, or audit trails are accessible only to authorized systems.

Beyond document tracking, WRCode could extend to physical or virtual access control. For example, access to a room, workstation, or digital workspace could be bound to a session, with logs recording who entered, who viewed a document or media file, and when. These events would be anchored on the IOTA ledger, providing immutable timestamps. The orchestrator could then visualize these anchored events as part of the WRCode context, giving users and auditors a clear historical record without embedding the raw timestamps directly in the WRCode itself. The integration of customizable multi-AI agent workflows makes this concept particularly powerful: agents could not only monitor and enforce access, but also orchestrate actions such as contextual policy adjustments, automated reporting, or mandatory encryption of all session outputs at closure.

In addition, all outbound data flows could be scanned at the device boundary. If WRStamped content is detected leaving the device through unauthorized channels, the system would enforce a fail-close response—blocking the transfer instantly and generating a signed report with details such as channel, session, and timestamp. These events could then trigger automated workflows, ranging from user notifications to device isolation, depending on context.

This feature may not be available in the initial release, but it highlights how WRCode can evolve from pure policy enforcement towards a groundbreaking framework that unifies prevention, accountability, access control, outbound monitoring, encryption-in-transit, IOTA-anchored auditing, embedded-code watermarking, and customizable AI orchestration.

Seamless Session Management and Cross-Device Identity Syncing with WRCode



Seamless Session Management and Cross-Device Identity Syncing with WRCode

Traditional authentication systems suffer from fragmented identity handling, insecure session storage, and limited interoperability across devices. WRCode introduces a new approach to session management and device authentication by allowing users to cryptographically prove their identity using a scanned QR code. This paper outlines how WRCode can be used to establish persistent, cross-device, privacy-preserving sessions without sharing sensitive data.

2. Login and Identity Binding with WRCode

WRCode provides a login flow where a user authenticates by scanning a QR code displayed on a trusted website or device. The key components of this flow are:

- Session-anchored identity: A WRCode ties the session to a verifiable, account-bound identity anchored on Solana.
- Zero-knowledge proof exchange: Instead of revealing credentials, the user presents cryptographic proof of session ownership.
- No password transmission: All validation is based on signed session tokens and local identity proofs.

3. Cross-Device Syncing and Persistent Sessions

Once a WRCode login is completed on one device (e.g., a smartphone), a sync token can be issued and paired with other devices. These devices can then:

- Share context and session state without exposing PII
- Trigger commands or automation workflows remotely
- Operate the same orchestrator instance (e.g., WR CODE) from different devices

This system removes the need for conventional OAuth logins, cookies, or 3rd-party trackers while maintaining full continuity of experience.

4. Secure Voice-Driven Orchestration via Synced Devices

Once identity is synchronized, the user can issue commands from a mobile device such as:

"Schedule a call with Anna this Friday"

The voice agent:

- Authenticates against the synced orchestrator session
- Applies access control policies and context awareness

- Triggers AI workflows securely, without re-authentication

This makes WR CODE-based orchestration both voice-activated and context-aware, with strong session integrity.

5. Revocation and Token Expiry

WRCode session tokens follow a strict security mechanism:

- Each token is account-bound, linked to a verified WRCode.org identity.
- Tokens are hardware-bound, incorporating device-specific entropy to prevent duplication or misuse.
- Tokens can be revoked at any time from within the user's WRCode.org account, offering full control in case of device loss or compromise.

To ensure secure session lifecycle management:

- Each WRCode session token has a configurable TTL (time-to-live)
- Users may revoke sessions manually via WRCode.org
- Tokens are bound to device-specific hashes, preventing reuse on unknown hardware

6. Benefits Overview

| Feature | Traditional Logins | WRCode Identity Flow |
|-------------------------------------|--------------------|------------------------|
| Password required | ✗ Yes | ✓ No |
| Third-party cookie/session tracking | ✗ Often used | ✓ None |
| Cross-device session control | ✗ Fragmented | ✓ Synced and secure |
| Privacy-preserving | ✗ Low | ✓ Zero knowledge |
| Revocation granularity | ✗ Coarse | ✓ Per-device, realtime |

7. Conclusion

WRCode's session and identity model simplifies login and session syncing across devices while elevating privacy and security. This mechanism makes it possible to run a single secure orchestrator session across multiple devices, enable zero-trust orchestration commands from mobile agents, and eliminate traditional authentication flaws like password reuse and session hijacking.

By anchoring session proofs and identity bindings cryptographically and combining this with AI agent control logic, WRCode enables an entirely new model of fluid, secure, and privacy-first interaction.

The WR CODE orchestrator is fully usable without login for general automation tasks, but all sensitive features—such as secure payments via Vault, trusted email, or encrypted autofill—are restricted to its security layer. This layer is activated through user-defined methods like WRCode scan, autopairing (e.g., when the smartphone and orchestrator are in the same local network), voice command, hardware key, or biometric authentication.

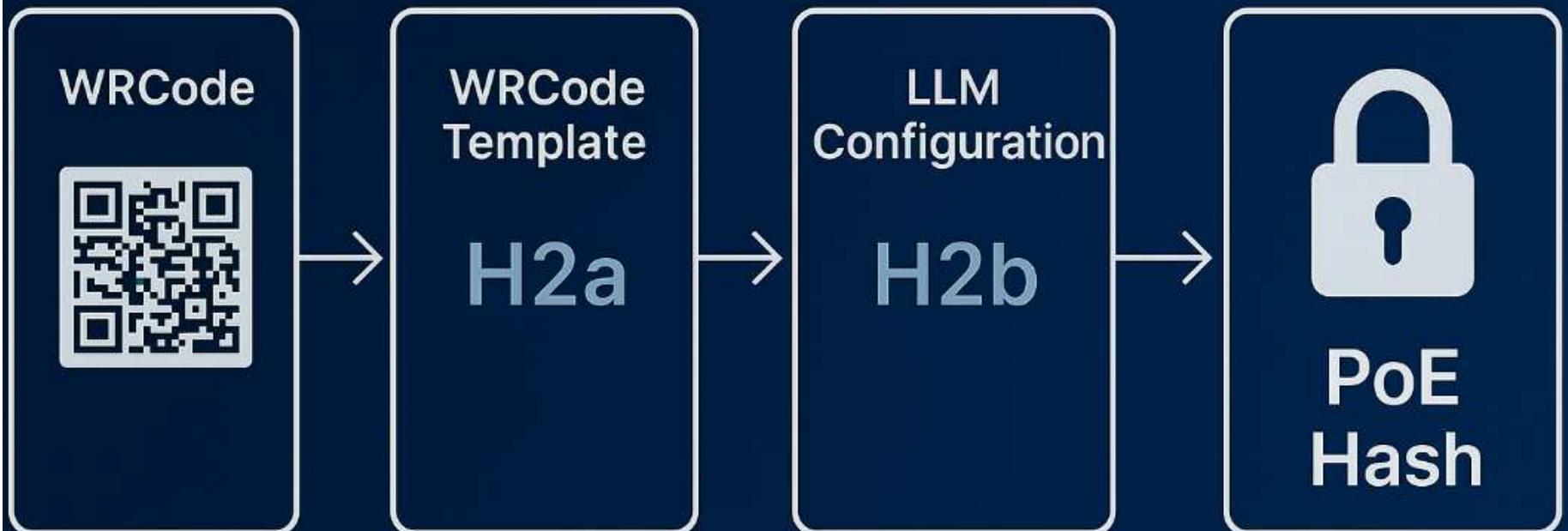
Once access is granted, a secure session token is issued. This token is cryptographically bound to the user's hardware, software instance, and verified WRCode.org account. It cannot be reused outside its original context, making it effectively useless if stolen. These tokens are fully revocable at any time via the user's WRCode.org account, offering a strong control mechanism in case of compromise or device loss.

When logged into the secure layer, the orchestrator can automatically log the user into websites, and even perform signups, depending on their configured preferences. Sites that support WRCode login will recognize the active session and allow instant access. For traditional websites without this support, WR CODE uses a local password manager—secured within the same environment—to handle login credentials automatically and securely.

This layered design ensures a frictionless yet secure user experience, balancing convenience with robust protections. Default configurations remain conservative, but high-security options like MFA, fingerprint unlock, and fast logout are available for users with elevated risk profiles.

Proof-of-Execution

WRCode templates combined with AI and cryptographic verification



Proof of Execution (PoE) — Trustless Verification of AI Execution

As AI automation systems increasingly guide decisions in regulated, high-impact domains—such as finance, law, healthcare, logistics, and enterprise optimization—the question of trust and accountability becomes central. Users, regulators, and stakeholders must be able to verify not only the results of AI-augmented workflows, but also the integrity of the processes that produced them.

Proof of Execution (PoE) is a cryptographic integrity layer embedded in the WRCode trust framework. It ensures that any automation or optimization triggered by a WRCode is verifiably authentic, tamper-evident, and auditable. By anchoring inputs, models, templates, and outputs in a transparent proof chain, PoE enables predictable, reproducible, and compliance-aligned AI automation across devices and environments.

Companies that offer WR Codes as part of their services must ensure that their AI orchestration environment are configured in a tamper-resistant, auditable way. This enables outcomes to become more predictable and aligned with predefined compliance goals, reducing risk and increasing the reliability of WRCode-triggered automations.

PoE is not just a technical mechanism—it reflects a design philosophy: automation must remain transparent, accountable, and human-verifiable, even as AI systems become more capable than the people using them. This architecture preserves oversight, facilitates auditability, and helps build long-term trust in decentralized intelligent systems.

By default, all WRCode executions are strictly **local-only**. This applies both to the execution environment and the use of models. Mobile devices and lightweight orchestrators are designed to process WRCode logic entirely on-device, without calling external services. No data is transmitted unless the user explicitly opts in.

Most WRCode-compatible applications are expected to support a **dual execution mode**: a local mode (enabled by default) and an advanced orchestration mode, which can optionally enable cloud-based AI processing. Notably, **dual mode can still be entirely local** —for example, advanced orchestration on a powerful edge device may use multiple local models with no external connectivity. However, some advanced WRCode templates may recommend enabling cloud inference for enhanced reasoning. This mode remains strictly optional and must be explicitly toggled by the user. Execution always remains under full user control. No WRCode template will ever initiate external cloud AI usage by default—**cloud execution must be explicitly toggled on** by the user, even in advanced orchestrator setups. This ensures that no WRCode workflow can silently escalate to cloud-based inference without full awareness and consent.

This separation also simplifies PoE, as local-only inference offers stronger and more verifiable protections. PoE provides a cryptographic receipt of the session—anchoring the declared input, reasoning context, and output trace in a tamper-proof, decentralized manner.

Importantly, PoE is **not a mechanism for deterministic output reproduction**, especially when using LLMs. Instead, it serves to prove:

- What WRCode was scanned
- Which templates and context were executed
- Who published them
- Under what declared model and orchestration configuration the reasoning occurred
- What output was generated at that specific point in time

PoE Captures the Orchestration Snapshot

1. Input and Template Integrity

- The scanned WRCode payload and its Project ID
- All associated TemplateIDs, hash-verified and published under that Project
- All templates must be public, immutable, and match their hash on wrcode.org

2. Publisher Verification

- Pro Users can publish community templates after verifying their email

- WRCode Publishers (required for active WRCode deployment) must prove domain ownership or equivalent organizational control (e.g., DNS TXT record)
- The publisher's verified identity is permanently embedded in the PoE trace, and is confirmed using zero-knowledge proofs via cryptographic anchoring—ensuring that no PII is ever exposed during identity verification or orchestration logging. Zero-knowledge proofs are considered a best practice for privacy-preserving attestations in the context of GDPR and upcoming AI governance requirements (e.g., EU AI Act Art. 14 and 23). Where used, ZKPs provide verifiable records of user consent, session context, or policy compliance — without exposing personal data. ZKP methods are optionally supported via third-party libraries, following established standards (e.g., zk-SNARKs, W3C Verifiable Credentials). No original ZKP scheme is proposed. WR CODE is a browser-based orchestration framework supporting optional cryptographic primitives such as zero-knowledge proofs, secure enclaves, and timestamp anchoring. Core features are built on source-available components. Advanced mechanisms are pluggable and depend on the security requirements of each use case. WR CODE supports modular integrations of established privacy-preserving techniques. Where required (e.g., under GDPR or AI Act compliance contexts), users may implement zero-knowledge proof flows using standard open source protocols such as zk-SNARKs (via libsnark or circom), Bulletproofs, or verifiable credentials via Hyperledger Aries (Idemix). These are not required for core

system functionality and are left to the user's discretion based on use case and trust requirements.

- WR CODE itself does not introduce novel cryptographic primitives. All optional ZKP integrations rely on public, licensed cryptographic frameworks, and are compatible with privacy-first design principles.

3. Model and Reasoning Context (Declared Runtime)

- Declared model provider (e.g., local, OpenAI, Claude)
- Declared model version or name (if known)
- Model parameters (temperature, top-p, max tokens, etc.)
- Context window size and memory constraints
- WRVault access log (critical/sensitive/normal) — vault presence is recorded, not its contents
- Publishers must also attach a plain-text declaration file (e.g., runtime-config.json) to their projects. This file contains the declared model provider, version, parameters, and environment under which the publisher conducted their PoE.

We focus here on the practical integration of PoE into two core inventions: **WRCode-triggered workflows** and the **WR CODE orchestrator**, which together enable **real-time AI optimization processes and complex orchestrated AI automation processes** with optional human input. While we do not claim that cryptographic verification of orchestrated AI processes is entirely new, the specific combination and application of these elements is a functional and strategic innovation we describe here in a **descriptive and implementation-focused** manner.

Importantly, PoE also prepares for a future in which AI systems may become more capable than humans in certain reasoning tasks. In such cases, maintaining human oversight becomes essential—not only for ethical reasons, but for governance, accountability, and societal trust. A cryptographic anchor like PoE ensures that even when humans defer to AI-driven suggestions, they retain a clear, verifiable record of how those decisions were derived.

README.md

A short **README** must be included to explain how the application or workflow behaves from a user perspective. It should provide a clear, non-technical overview of what the WRCode project does, how the automation behaves, and what the user can expect during execution. This ensures transparency and usability, and helps build trust by making PoE-backed automations understandable to all users.

Example README.md – Transit WRCode App (User-Focused)

markdown

Transit Companion: WRCode-Powered Routing Assistant

This WRCode project enables users to quickly access route planning and disruption assistance using secure, locally executed AI workflows. The experience is triggered by scanning a WRCode at supported transit stations, signage, or printed schedules.

📱 What Happens When You Scan the Code?

1. The app opens and asks where you want to go.
2. You type in a destination.
3. The system finds the best route based on live schedules and stored data.
4. If there's a disruption (e.g. delays, cancellations), it may suggest:
 - Alternative transport
 - Nearby coffee or break options
 - Context-aware info like platform changes

🔒 How Is Your Data Protected?

- All logic runs **locally** on your phone by default.
- Your preferences and behavior are stored in a secure personal vault (WRVault).
- Nothing is sent to the internet unless you **manually opt into advanced optimization mode using cloud-based AI like GPT4**.

📄 What's Inside This WRCode?

- A routing template (defines how travel logic works)
- Optional optimization templates (suggestions if disruptions occur)
- A configuration file explaining how the system behaves
- A PoE (Proof of Execution) that verifies the whole process is secure and verifiable

📄 What Is Proof of Execution (PoE)?

PoE is a tamper-proof receipt that shows:

- What was scanned and used
- Who is the publisher
- Which AI models helped generate the results

- What the system suggested at the time of PoE
- That no part of the experience was altered or misused

Users can download their own `PoE.json` file as proof—useful in audits, logs, or future reference.

🔧 Technical Requirements

- A WRCode-scanner
- A modern smartphone (recommended 8GB+ RAM)
- Optional: advanced orchestration can be toggled on by the user for enhanced reasoning

🗃 User Tip

Look for WRCode signs at stations or printed materials. WRCodes require an internet connection for integrity checks in order to verify the unaltered status of PoE. This makes sure that wrcodes are secure, protecting the users from any potential risk. For more infos visit wrcode.org

template-map.json — Mode-Specific Template Mapping

The template-map.json is a **mandatory, tamper-proof configuration file** downloaded automatically from wrcode.org during every WRCode scan. It plays a critical role in defining which automation templates are used in different execution modes:

- default: templates intended for **mobile or lightweight local environments** (typically just one minimal template).
- orchestrated: a list of **multi-agent templates** used in advanced or cloud-assisted scenarios.

This file ensures that each execution context is **predictable, transparent, and cryptographically bound** to its published orchestration logic. It is publicly auditable, hash-verified, and bundled with the project's official WRCode templates. Including it in the PoE chain prevents tampering and makes template resolution verifiable across devices and environments.

The template-map.json defines not only which templates are activated per execution mode, but also how each template should present its output—either silently in the background or visibly prioritized in numbered UI slots—allowing the orchestrator to control user-facing display behavior in a transparent and verifiable way.

In more advanced WRCode orchestrations, the relationships between templates are defined using a **pointer graph**. This graph specifies how execution flows between templates, including roles such as

supervisors, coordinators, or context evaluators. These links can be expressed through a separate orchestration-graph.json file or embedded within template-map.json, allowing the orchestrator to resolve execution dependencies and priorities in a traceable, auditable way.

Example: template-map.json (Loaded in Runtime)

```
{  
  "mode_templates": {  
    "default": {  
      "template_ids": [1257],  
      "display_modes": {  
        "1257": "visible"  
      }  
,  
      "orchestrated": {
```

```
"template_ids": [4278, 4547, 4518, 4519],  
"display_modes": {  
    "4278": "visible:1",  
    "4547": "silent",  
    "4518": "visible:2",  
    "4519": "visible:3"  
},  
"pointer_graph": {  
    "template": 4278,  
    "4278": {  
        "role": "input coordinator",  
        "next template": [4547]  
    },  
    "4547": {
```

```
        "role": "supervisor",
        "next templates": [4518, 4519]
    },
    "4518": {
        "role": "helper 1",
        "next template": [4519]
    },
    "4519": {
        "role": "output coordinator",
        "next": []
    }
}
```

,

This pointer graph enables complex multi-agent orchestration logic to be described, audited, and verified as part of the PoE framework.

Terminology within the pointer graph carries functional significance:

- **Input Coordinator:** Monitors input streams and decides when to initiate delegation.
- **Supervisor:** Coordinates the reasoning and task assignment to downstream helpers.
- **Helper 1–n:** Perform specific, well-defined subtasks, often in parallel or under supervisor control.
- **Output Coordinator:** Consolidates the final result and handles the display or user-facing routing.

This file is:

- **Mandatory** and downloaded automatically during WRCode scan.
- **Published transparently** on wrcode.org with the associated project.
- **Hash-anchored** in the PoE to ensure integrity and verifiability.
- Used by the orchestrator to load the appropriate logic tree depending on execution mode.

Each role helps structure the orchestration process in a transparent and audit-ready manner, and may be reflected in display priority and execution order.

📁 AI Template — Default Example, Mobile only, local LLM (Loaded in Runtime)

Template 1257 – Basic Transit Routing (for mobile default mode)

json

```
{  
  "template_id": 1257,  
  "name": "Basic Transit Routing",  
  "version": "1.0",  
  "purpose": "Determine route from current location to user-entered destination",  
  "triggers": [  
    "wr_code_scan"  
  ],
```

```
"inputs": [  
    "user_input_destination",  
    "geolocation_current_position"  
,  
  "actions": [  
    {  
      "type": "route_lookup",  
      "engine": "offline_gtfs",  
      "output": "route_summary"  
    },  
    {  
      "type": "display_result",  
      "format": "simple_card",  
      "content": "{{route_summary}}"
```

```
}
```

```
],
```

```
"context_mode": "local",
```

```
"output_trace": true
```

```
}
```

 AI Template — Orchestrator Example (For simplification only 1) Loaded in Runtime

Template 4518 – Platform Reassignment Assistant (for orchestrated use)

json

KopierenBearbeiten

```
{
```

```
"template_id": 4518,
```

```
"name": "Platform Reassignment Assistant",
```

```
"version": "1.0",
```

```
"purpose": "Guide user when platform change is detected",
"triggers": [
    "platform_changed",
    "train_delay_detected"
],
"inputs": [
    "user_current_station",
    "train_id",
    "new_platform_info"
],
"actions": [
{
    "type": "notify_user",
    "method": "context_card",
```

```
"message": "Platform changed to {{new_platform_info.platform}} for train {{train_id}}."  
},  
{  
    "type": "suggest_followup",  
    "options": [  
        "Show route from current location to new platform",  
        "Notify other affected travelers"  
    ]  
}  
],  
{"context_mode": "hybrid",  
 "orchestration_tags": ["disruption_management"],  
 }  
}
```

📁 LLM Configuration Template — Recommended LLM Configuration (PoE Evidence)

The recommended configuration is included as part of the PoE chain and must be cryptographically anchored by the publisher. It must contain a baseline configuration for mobile local-only execution and may optionally include advanced orchestration settings for multi-agent setups. This enforces adherence to the originally declared technical conditions for any execution claiming to represent a specific WRCode project, providing a transparent and verifiable baseline for both minimal and advanced deployments.

📄 Config example for mobile, local only (default):

```
"mode_templates": {  
    "default": [557],  
    "orchestrated": [418, 419, 420, 421]  
},  
{  
    "mode": "default",  
    "llms": [  
        {  
            "name": "Hugging Face Model",  
            "model_id": "Hugging Face Model ID",  
            "model_type": "text-generation",  
            "model_version": "v1.0",  
            "model_status": "active",  
            "model_desc": "A large language model trained on a diverse dataset.",  
            "model_params": {  
                "max_length": 512,  
                "temperature": 0.7,  
                "top_p": 0.9,  
                "top_k": 50  
            }  
        },  
        {  
            "name": "Custom Model",  
            "model_id": "Custom Model ID",  
            "model_type": "text-generation",  
            "model_version": "v1.0",  
            "model_status": "active",  
            "model_desc": "A custom-trained language model for specific tasks.",  
            "model_params": {  
                "max_length": 512,  
                "temperature": 0.5,  
                "top_p": 0.95,  
                "top_k": 40  
            }  
        }  
    ]  
}
```

```
{  
    "model_provider": "TinyLlama",  
    "model_version": "1.1B-GGUF-Q4_0",  
    "purpose": "retrieval"  
},  
  
{  
    "model_provider": "Mistral",  
    "model_version": "7B-Instruct-Q4_0",  
    "purpose": "reasoning/summarization"  
}  
,  
{"used_on": "2025-07-14T12:00:00Z",  
 "notes": "Mobile-only WRVault configuration for local execution. No cloud interaction. Devices support dual LLMs—TinyLlama for retrieval and Mistral 7B for reasoning."}
```

```
    "environment": "local"  
}
```

📄 Config Example for advanced multi-agent orchestration:

```
{  
  "mode": "orchestrated",  
  "llms": [  
    {  
      "model_provider": "Mistral",  
      "model_version": "7B-Instruct-Q4_0",  
      "purpose": "local pre-filtering"  
    },  
    {  
      "model_provider": "OpenAI",  
      "model_version": "text-davinci-003",  
      "purpose": "global filtering"  
    }  
  ]  
}
```

```
        "model_version": "gpt-4",
        "temperature": 0.9,
        "purpose": "semantic reasoning"
    },
],
"combined": true,
"used_on": "2025-07-14T12:10:00Z",
"notes": "Hybrid setup using local Mistral for contextual refinement and OpenAI GPT-4 for cloud-based inference, coordinated by an advanced orchestrator.",
"environment": "hybrid"
}
```

These files are all cryptographically anchored cryptographically by the publisher and published on wricode.org alongside the ProjectID. It serves as a transparent, tamper-evident reference. While this configuration cannot enforce cloud model behavior, it proves the declared setup used during PoE. In

cases where external cloud AI is involved, this approach helps approximate the execution context and narrows the reproducibility boundary.

Example Session Trace

Publishers need to provide a reproducible example session file (e.g., `example-session.json`) alongside the `runtime-config.json`. This file illustrates a test case under declared conditions—without exposing real user data.

Example structure:

```
{  
  "system_prompt": "Where do you want to go?",  
  "user_input": "Berlin Central Station",  
  "context_summary": "live routing status for geofencing in Berlin Zoo; optimization layer enabled to interpret changes in real-time, such as train platform updates",  
  "output_observed": "Please proceed to platform S3. A train departs to Berlin Central Station every 5 minutes.",
```

```
"output_hash": "bd83fe8a... (truncated)",  
"linked_proof_of_orchestration": "abc123-PoE"  
}
```

This allows users and auditors to verify the intended template behavior using neutral, anonymized examples. These files are not used for live execution, but serve as reproducible semantic baselines. They enhance trust without requiring access to user-specific or PII-containing content.

4. Orchestration Setup

- Which templates were activated and in what order
- Pointer graph and routing logic between template steps
- Display logic: e.g., silent execution, user prompt, UI integration
- Sandboxed execution constraints (RAM-only, read-only runtime)

5. Output and Reasoning Trace

- The actual output may vary across identical inputs, due to the probabilistic nature of LLMs

- Even if inputs and model parameters are identical, LLMs may generate slightly different completions
- PoE includes a snapshot of the output generated during the session, with a mandatory content hash to serve as proof
- While not deterministic, this proves the output was created under verifiable conditions
- PoE thus provides forensic traceability and narrows the output likelihood space

🛡 Enforcement Rules and Identity

- Only published templates with matching hashes under a valid ProjectID will be executed in WRCode scanners
- Unpublished or modified public templates or context data stop functioning by design
- Modified templates must be republished under a new ProjectID by the user
- Optimization layers (post-WRCode, user-defined logic) are normally excluded from PoE and are intended as a customizable feature for users. However, WRCode publishers may optionally provide optimization templates as part of their published projects. If included, these templates are treated like any other: they must be registered, verified, and publicly declared. Registered

optimization templates are then included in the PoE scope. Regardless of inclusion, optimization layers are always toggled off by default and must be explicitly enabled by the user. Once activated, these templates can dynamically respond to session context—such as detecting that a train was cancelled or that wait time has exceeded a threshold—allowing adaptive, privacy-aware behavior while respecting the user's control and configuration.

To further extend flexibility, WRCode providers may optionally publish dedicated optimization templates, which are designed specifically for use in the optimization layer. These templates are not triggered by the WRCode scan directly, but act as supporting workflows that can be invoked by the optimization logic when the user's input or situation requires adaptive handling. This allows WRVaults to be extended with real-time intelligence across edge or orchestrator environments—ensuring the user remains fully prepared across all relevant scenarios.

Example optimization template (opt-confused-routing.json):

```
{  
  "template_id": "opt-confused-routing",  
  "purpose": "respond to public transport service disruption or extended waiting time while  
  protecting location privacy",  
  "triggers": [  
    ...  
  ]  
}
```

```
"train cancelled",
"route interrupted",
"platform not assigned",
"wait time exceeds 30 minutes"

],
"actions": [
{
  "type": "evaluate_context",
  "input": "last known location",
  "obfuscation": "enabled"
},
{
  "type": "suggest_activities",
  "options": [
```

```
"Visit Café Altona (recommended)",  
"Grab lunch at your preferred local spot",  
"Explore Altonaer Balkon park nearby"  
,  
  "decision_logic": "based on time of day, WRVault user preference memory, and local radius"  
,  
{  
  "type": "generate_instruction",  
  "prompt": "Offer a friendly suggestion for using the delay time. Balance usefulness with privacy."  
,  
  "context_mode": "local",  
  "confusion_policy": "tree-of-confusion",
```

"optional_notes": "If a long wait is detected, this template checks past user behavior and preferred places from WRVault memory and suggests nearby options without exposing precise location."

}

This template would not execute on scan but would be available in the optimization layer if toggled on by the user. Publisher identity is strictly enforced for WRCode Publishers via domain control; Pro users have email-level identity only

User Roles and Trust Boundaries

- Standard Users: scan WR Codes, inspect inputs, verify outputs
- Pro Users: email-verified, can create and share community templates
- WRCode Publishers: domain-verified, may publish active ProjectIDs for public WRCode execution
- PoE embeds these identities to enable full traceability and prevent impersonation or silent tampering.

Practical Hashing and Trust Model in PoE

| Data Element | Can Be Hashed? | Trust Method |
|--------------------------------------|----------------|--|
| WRCode Payload, ProjectID | ✓ | Cryptographic hash |
| Template Files (JSON) | ✓ | Full hash + Solana anchoring |
| Publisher Identity | ✓ | Domain proof + Solana zero-knowledge |
| Declared Model Settings | ✓ | Included in signed manifest |
| Actual Cloud Inference (e.g., GPT-4) | ✗ | Cannot verify; declaration only |
| Output Text | ✓ (trace only) | Snapshot hash, not reproducible |
| Optimization Layer | ✗ | Disabled by default; excluded unless published and need to be toggled on per default |

* PoE Capabilities and Limitations

| Aspect | Verified by PoE? | Notes |
|-----------------------------------|------------------|---|
| WRCode Payload | ✓ | Fully hashed and anchored |
| Templates + ProjectID | ✓ | Must match published hashes |
| Publisher Identity | ✓ | Domain-verified for publishers |
| Declared Model + Parameters | ✓ (Declared) | Logged by caller, not enforced by cloud model |
| Actual Cloud LLM Runtime Behavior | ✗ | Not directly measurable unless local |
| Output | ✓ (Traceable) | Can vary, but trace can be hashed |
| Optimization Layer Behavior | ✗ | Out of PoE scope |

PoE is a cryptographically anchored receipt, not a replay mechanism. It helps verify the input state and declared configuration that led to a specific session and output. For sensitive or regulated use cases, PoE ensures that automations are accountable—even if the language model's output varies slightly across sessions.

This design provides strong protections for public, tamper-proof, and identity-bound orchestration—while remaining realistic about the nondeterministic nature of large-scale AI models.

1. The Need for Verifiable AI Suggestions in complex orchestrated workflows

- While many orchestration frameworks allow AI agents to process, synthesize, and suggest optimized actions, few provide any assurance that these outputs:
- Followed a consistent set of logic or business rules;
- Used approved AI models, orchestration templates and context;
- Have not been tampered with in-flight;
- Can be externally audited.

This is especially problematic in sectors where AI is used to assist:

- Financial recommendations or contract generation;
- Legal intake forms or eligibility assessments;
- Safety-critical procedures or compliance workflows.
- critical decision-making in enterprise or organizational environments

Without a clear verification layer, even well-intentioned AI assistance can be dismissed as opaque or risky. **WRCode** templates, when combined with the specific configuration of the **WR CODE** orchestrator—including workflow templates, user-defined settings, and AI model setup—can be cryptographically sealed using the PoE approach. This tamper-proof linkage ensures that the resulting optimization is reproducible, narrowing down the variability of AI outcomes in defined contexts. In such scenarios, the result becomes less random and more consistent with organizational intent or regulatory expectations.

2. The Structure of a Proof of Execution

A valid PoE consists of the following components:

Structure of a Proof of Execution (PoE)

| Component | Description | Hash ID / Trust Layer |
|-----------------|------------------------------------|------------------------------------|
| Input Context | Original user input or task prompt | $H1 = \text{SHA256}(\text{input})$ |
| Runtime Session | | |

| Component | Description | Hash ID / Trust Layer |
|-------------------|---|--|
| WRCode Template | Static WRCode configuration (QR-encoded schema) | H2a = SHA256(wrcode_template) |
| Template Logic | All referenced orchestration templates (mobile + orchestrated) | H2b = SHA256(template_files[]) |
| Pointer Graph | Optional routing graph between templates (e.g., 4278 → 4547 → 4519) | Embedded in template hash / PoE trace |
| Display Logic | Display modes and slot priorities (e.g., visible:1, silent) | Part of template-map.json, hash included |
| LLM Configuration | Model provider, version, parameters, environment | H3 = SHA256(runtime-config.json) |
| Reasoning Output | AI-generated suggestion or action trace | H4 = SHA256(output) |
| Publisher Proof | Domain-verified ID for WRCode publishers via DNS or smart proof (ZKP over Solana) | Solana Anchored Identity |

| Component | Description | Hash ID / Trust Layer |
|--------------------------|--|--|
| Pro User Proof | Email-verified community template creator | Signed + logged under PoE |
| Template Map | Mandatory file defining active templates per mode and display order | SHA256(template-map.json) |
| Mobile AI Templates | Default logic for local execution (e.g., ID 1257) | Covered in H2b, included in map hash |
| Multi-Agent AI Templates | Orchestrated templates with role assignments and routing graph | Covered in H2b, verified via map+graph |
| README | Required human-readable description of workflow behavior | Included in project hash suite |
| Final Proof Hash | Combined signature: $H = \text{SHA256}(H1 + H2a + H2b + H3 + H4 + H5)$ | Anchored on Solana or local ledger |

The system may optionally embed:

- Timestamps

- Session IDs
- Role-based execution context
- Privacy markers (e.g. PII masking level), obfuscation techniques, decoy reasoning paths, cryptographic math checksums, reasoning chain splitting, and distributed workload distribution
- Embedded context into LLMs (e.g. injected memory or prior interactions)
- Configuration templates provided by WR Code publishers (e.g. a WR Code generator on a website that embeds a customizable wishlist, enabling geofenced or personalized experiences)

All of which can remain local or be anchored (as hashes only) on distributed ledgers such as Solana, enabling tamper-evident proofs without data leakage.

3. Workflow Integration

Within the WR CODE orchestration system, PoE can be integrated into:

- WR Code workflows
- AR/VR automation triggers
- Mobile app suggestions
- NPC or robotic system automation triggers

- Token-bound NFT actions for user or agent state transitions
- Compliance-oriented exports (e.g., audit logs, signed reports)

What makes this architecture distinct is the combination of **QR-based WR Codes** with **real-time AI optimization logic and ai automation workflows **. A user can scan a WR Code to trigger a locally stored or remotely verified workflow template loaded in runtime, which is then processed by **WR CODE** with optional human interaction and AI agents. The result—an optimized suggestion, recommendation, or output—is then documented cryptographically using the **PoE** format.

A `.PoE.json` file is automatically generated and includes the full hash chain, a human-readable summary, and all referenced templates and configuration files. These hashes are **mandatory and anchored cryptographically**, forming the cryptographic backbone of WRCode verification. Without a valid **PoE**, no **WRCode**-based execution is permitted.

WRCode applications **require an active internet connection** to validate the hashes in real-time against public records on `wrcode.org`. If no connection is available, the system falls back into QR-only mode, where execution is disabled. Even in this fallback state, only **locally cached, whitelisted WRCode publishers** are allowed to load their WR templates.

The **PoE** artifact can:

- Serve as evidence in compliance audits
- Be submitted as a zero-knowledge credential

- Trigger smart contracts tied to verified outcomes

4. Example Use Case: WRCode-Triggered Business Onboarding Flow

A WRCode provider publishes a QR-based code that encapsulates a geofenced business onboarding flow. When scanned at a retail kiosk, this WRCode launches an WR CODE orchestration sequence customized to the visitor's preferences (e.g., loyalty tier, language, or location-specific offers). The WRCode references a cryptographic token anchored on Solana that proves the existence and integrity of the associated template, which defines the automation steps and may embed context where appropriate, enabling consistent and reproducible execution while allowing tailored runtime personalization when needed. The orchestrator fetches the referenced WRCode template and injects it into a local or hybrid LLM. While the setup on the mobile device and orchestrator is controlled by the user, the WRCode provider can supply a verified process logic, ensuring the user experience remains consistent. This approach allows the provider to unify outcomes across devices while

5. Strategic Implications

Proof of Execution is not a product, but a design pattern:

- Compatible with decentralized automation
- Extendable to zero-knowledge proof systems

- Aligned with GDPR, and compliance principles
- Enabling fair automation, AI transparency, and anti-bias controls

It can be used internally, locally, or in cross-organizational setups. Its cryptographic roots ensure that even future audits (e.g. post-quantum) can validate past decisions.

6. Integration with WR Codes and NFTs

By combining PoE with:

- WR Codes (QR + Solana): verifiable workflow triggers that may contain plaintext AI logic, execution templates, or preconfigured orchestration instructions ERC-6551 NFTs : action logs tied to digital identity

... WR CODE can offer **end-to-end traceability** — from user input, through AI optimization, to decentralized action. PoE adds an **execution-time integrity layer** to this setup, providing real-world verifiability for each decision.

7. Closing Note

Proof of Execution is a natural extension of Optimando's architecture: privacy-first, audit-capable, modular, and locally executable. It offers a clear response to the growing demand for **explainable, traceable AI** in professional settings.

While further research is needed for formal ZKP extensions or tokenized enforcement, PoE provides a practical foundation — especially in WR Code-powered orchestration flows.

We do not claim novelty of the PoE mechanism in a legal or academic sense; instead, we present this as a **pragmatic approach** to enhancing trust and transparency within our open architecture.

While QR code–triggered AI workflows have appeared in recent commercial and academic prototypes, they generally do not provide tamper-proof audit logs, configuration verification, or standardized vaults. The architecture described here introduces a verifiable, cryptographically anchored **vault layer**, which sets it apart by enabling reproducibility, compliance-grade proof structures, and decentralized orchestration transparency.

WR CODE: Real-Time AI Orchestration for Mobile, AR, Robots, and Desktop—With Predictive, Privacy-Preserving Automation

WR CODE is a modular AI orchestration framework that delivers real-time, context-driven automation across desktop workstations, mobile devices, AR headsets, and even robots—while maintaining privacy, human oversight, and local data control.

At its core is the WR CODE Orchestrator, a desktop-based control layer that holds the company's entire knowledge base—including technical documentation, compliance checklists, and best practices—ensuring that every user, device, or agent receives consistent, expert-level guidance.

How It Works:

1. The **orchestrator** runs on secure local infrastructure, coordinating AI-driven workflows based on:
 - **User role**
 - **Global task intent and ai agent logic (predefined by templates)**
 - **Real-time contextual inputs** (e.g., geolocation, ZKP-verifications, environmental signals, sensor data, vision data, audio data and other input data)
2. Outputs are streamed to:
 -  **Desktop displays / VR Devices**
 -  **Mobile devices**
 -  **AR overlays**
 -  **Robots or smart devices**

The backend triggers **AI-powered workflows** in real time:

-  An LLM agent interprets the user's role, context and task history
-  Predefined workflows or generated workflows are selected
-  Results are streamed back to the user's **smartphone or AR overlay** as visual information, task lists, alerts, or step-by-step instructions
-  Optionally, a session can include **local video/audio capture**, stored securely for documentation or post-processing—entirely within user-owned infrastructure
- The AR device or smartphone effectively acts as the **mobile master interface**, continuously collecting contextual input—including geofence events, GPS, user role, environmental data, speech commands, or text interactions. This input is securely looped to a **local or edge-based WR CODE orchestrator**, which interprets the intent and triggers predefined or dynamically generated automation workflows. The user does not need to manually initiate processes; instead, **presence and context alone** can activate backend logic. Optional **voice or text commands**—such as “*Show safety checklist*” or “*Draft report for Zone 3*”—can further refine or redirect the orchestration. Confirmations, overlays, or task results are streamed back in real time to the user's AR display or smartphone, enabling fully automated, hands-free interaction **without reliance on cloud infrastructure or invasive tracking**.

Example: Geolocation + Vision: Predictive, Context-Aware Support

- By combining **geolocation data** (e.g., GPS, geofence) with **visual inputs** (from cameras, AR devices, or sensors), the orchestrator can:
 - Accurately **understand the technician's environment** and **current task**.
 - **Predict likely next steps or possible risks and the most efficient approaches to fulfill the task.**
 - **Time the delivery of guidance** precisely—showing checklists, warnings, or support **exactly when needed**, without overwhelming the user with constant notifications. (AI Agents can adjust it dynamically)
 - For example:
 - A technician arrives at a site (**GPS triggers intent recognition, ZKF verified if needed**).
 - The AR glasses detect a specific machine type (**vision input**).
 - The orchestrator **automatically surfaces relevant repair steps, known issues, or safety warnings**—without being explicitly asked.
 - This **proactive assistance** is:
 - **Context-sensitive**
 - **Privacy-preserving** (no personal data leaves the infrastructure)
 - **Fully overridable or tunable** via **voice commands** or manual control.
-

Human, Robot, and Machine Collaboration:

- AR devices, smartphones, and robots can all act as **master tabs**—initiating or receiving tasks based on environmental detection or operator commands.
- Robots can contribute real-time data (e.g., location, vision, task state), helping the orchestrator to:
 - Predict needs before they arise
 - Trigger the right AI-driven workflow at the optimal moment
 - Loop in human supervisors when ambiguity or exceptions occur

Key Advantages:

Centralized Knowledge Repository:

All company knowledge is stored **locally**—keeping data safe and ensuring reliable, standardized support across the entire operation.

Predictive, Just-in-Time Automation:

By combining **geolocation** and **vision data**, the orchestrator can **anticipate task needs** and provide support at the right time—without unnecessary prompts.

Device-Independent Orchestration:

Desktops, mobile devices, AR glasses, and **robots** all work together in a **coordinated, flexible automation system**.

 **Privacy and Compliance by Design:**

Sensitive information never leaves the local environment. Optional **Zero-Knowledge Proofs** and **Solana anchoring** ensure **verifiability without exposure**.

 **Human-Centric:**

Automation assists—but never replaces—humans. The operator stays in control, with **support that can be refined, paused, or overridden at any moment**.

In short: **WR CODE** enables **context-driven, predictive AI orchestration** across people, devices, and machines—delivering **the right help at the right moment, powered by local knowledge, secured by design, and always under human control**. All actions can be logged locally, and the orchestrator can sync anonymized metadata or Zero-Knowledge-Proofs to a central server if needed—but raw data and personal context never leave the edge network .

Core Principles of WR CODE Geofence Automation (AR/Mobile Mode)

-  **Local ZKP Generation** – Presence is proven, but not revealed
-  **Edge-Synced AI Logic** – Smartphones and AR glasses connect to a secure local orchestrator
-  **Intent-Aware Workflows** – Role-, time-, and task-based logic selects appropriate responses

-  **No Tabs, No Dashboards** – UI is adapted for compact screens and overlays, local orchestrator runs the ai orchestration optionally guided by a backend operator
-  **Streaming Outputs** – Real-time instructions, summaries, or media rendered on mobile/AR interfaces
-  **Private Recording** – Sessions can be optionally logged (video/audio) under full local control
-  **Composable & Offline-Ready** – No dependency on public cloud or internet; logic runs on-site
-  **Verifiable, Auditable, Explainable** – Every input, activity, reasoning step, or geolocation event can be cryptographically anchored—optionally on a blockchain like Solana—and remains fully user-auditable. This ensures transparent, tamper-proof records without exposing personal data, enabling organizations to verify actions, context, and decisions at any time. This architecture makes **WR CODE** ideally suited for **in-the-field AI interaction**, **hands-free workflows**, and **privacy-first automation** across mobile and wearable form factors—whether in labs, warehouses, medical settings, or decentralized field operations.

Shops and zones can be whitelisted in advance, and geolocation-based detection can be toggled on or off at any time. Since only anonymous cryptographic hashes are generated locally not even at the

vendor site—without storing or sharing personal data—users remain fully in control and can decide when and where presence proofs are created.

Generalization Across Trigger Modalities

While this section focused on geofencing, the underlying principles of WR CODE—namely, **zero-knowledge validation of contextual triggers**, combined with **intent-aware AI orchestration**—are **technology-agnostic**. The same local verification and backend automation flow can be extended to a wide range of context-detection methods:

- BLE proximity beacons
- NFC or WR Code scans
- Wi-Fi SSID presence
- Camera-based zone detection
- Voice or gesture commands

Each of these modalities can serve as a secure, local trigger—optionally wrapped in a ZKP claim—enabling WR CODE to offer **privacy-preserving, explainable, real-time automation** across diverse environments such as industrial zones, smart offices, AR workspaces, or mobile field operations.

Just Another Possible Scenario: AI-Supported Alarm Systems

Using Optimando's architecture, users could configure a security setup where motion detectors trigger a live video stream analyzed by AI—capable of recognizing covered faces or visible weapons. In such cases, the system could escalate the incident, potentially even alerting authorities nearly in real-time. This scenario illustrates how WR CODE empowers users to orchestrate advanced AI workflows—extending far beyond the use cases covered in this paper.

Technical Considerations and Limitations

Some connected services may block automation, especially if login sessions expire or DOM manipulation is restricted. In such cases, the system provides fallbacks:

- Users manually log into relevant services in browser tabs
- Agents use visual or metadata-based cues to navigate and trigger content display
- Inaccessible features are flagged, and alternatives (e.g., via n8n, email relay, or webhooks) are proposed

The WR CODE architecture ensures reliability while retaining flexibility.

Optimized for High-Performance Workstations with Scalable Backend Flexibility

Optimando.ai is engineered to take full advantage of modern high-performance workstations — particularly those equipped with multi-core CPUs and high-end GPUs such as the NVIDIA RTX

5090. This setup supports fast, local execution of AI processes with high parallel throughput and minimal latency. It is especially well-suited for the demands of real-time, browser-based multi-agent orchestration.

Local Performance Capabilities

- **Local LLM Acceleration**

High-end GPUs enable efficient local inference of large language models such as LLaMA 3 or Mistral. A 5090-class GPU can process complex prompts with high token throughput and low latency, which is essential in multi-agent workflows where multiple reasoning paths run concurrently and interdependently.

- **Real-Time Speech-to-Text**

Transcription engines like Whisper (Large) benefit from GPU acceleration, enabling continuous, high-fidelity speech input to be processed in parallel with other tasks — critical for multimodal input scenarios and hands-free interactions.

- **Parallel Agent Execution**

With sufficient VRAM and compute power, multiple AI agents can operate simultaneously without affecting system responsiveness. This is particularly valuable in environments with multiple display slots, where agents generate, update, and refine results asynchronously.

Optional Automation Layer via n8n

While Optimando.ai is fully functional on its own, it can optionally be extended by integrating n8n as a backend automation engine — particularly useful for advanced or multi-layered workflow requirements. n8n is a separate system, but its modular, browser-based, and extensible design complements Optimando's orchestration principle perfectly.

When used together:

- WR CODE handles frontend orchestration and visualization of agent outputs,
- while n8n manages backend automation tasks such as:
 - multi-step validations,
 - scheduling,
 - agent chaining,
 - file processing,
 - external API interactions.

On powerful local hardware, n8n can run side-by-side with Optimando, offering low-latency automation while keeping full control over data flow and system logic.

Scalable Design for All Hardware Classes

For less powerful systems — such as laptops or entry-level desktops — n8n can be hosted on a dedicated self-hosted server. In this configuration, frontend devices act purely as lightweight visualization interfaces: they receive and render the output of AI agents without executing any heavy computation locally.

This architectural separation allows even modest hardware to participate in complex, distributed AI workflows without performance issues.

Importantly, all core components are self-hostable and source-available (or open source where noted):

- **WR CODE** (frontend orchestration layer, AGPL-3.0)
- **Local LLM backends** (e.g., Ollama / llama.cpp; tiny on-device models like TinyLlama, Phi-3-mini)
- **Vector databases** (e.g., Qdrant; Postgres + pgvector; **mobile**: SQLite + sqlite-vss or USearch/LanceDB for lightweight on-device indexing)
- **Automation layer** — lightweight defaults on PC/mobile (e.g., Node-RED [Apache-2.0], Huginn [MIT]); optional external companion **n8n** (source-available SUL)

While n8n is not open source in the OSI sense, it is licensed under a *source-available* model (Sustainable Use License). This means the full source code is publicly available, auditable, and modifiable for self-hosted use only — preserving transparency and enabling complete control over system behavior in professional environments. OpenGiraffe follows a similar source available license model OGPL 1.0 OpenGiraffe Proteted License.

Summary

The WR CODE architecture is capable of scaling from lightweight, local deployments to high-performance multi-agent orchestration environments — all while maintaining user transparency, data sovereignty, and full modularity. When combined with optional backend tools like n8n, it offers a powerful, extensible, and privacy-respecting alternative to centralized SaaS AI platforms.

Multi-Screen and VR Environments

The orchestration interface is built to take advantage of extended display setups and VR headsets. Multi-monitor workstations and VR browsers enable spatial separation of inputs, outputs, and visual control panels.

Skilled User Configuration

Although normal users can operate the system with preconfigured templates, its advanced features are best utilized by professionals with a background in:

- Local deployment of LLMs or AI tools
- Configuration of connector protocols like MCP, ChatGPT connectors, or custom APIs
- Orchestration of automation flows using systems such as n8n
- Understanding of browser-based control environments and DOM interaction constraints

Templates, aliases, and agent roles can be customized, but a working knowledge of automation frameworks and privacy-aware system design is recommended for full control.

Scalable and Modular by Design

The WR CODE architecture is inherently modular and scalable:

- **Entry-level users** can rely on hosted LLMs and run helper agents in minimal configurations.
- **Power users** can run multiple master tabs, integrate local and cloud LLMs, define trigger logic, and connect MCP-compatible services.

This flexibility ensures that the orchestration tool grows with its user base—from individuals testing workflows to full teams managing strategic operations.

In short, while accessible to a broad range of users, the system truly shines in professional hands, running on high-performance hardware, and configured by operators with domain-specific expertise in automation, AI integration, and real-time orchestration.

Technical Layer: Templates and Modularity

Each agent operates based on a **system prompt template** that defines its role, behavior, and expected output format. These templates are essential to task decomposition and relevance alignment. While templates are a core architectural element, their **quality and task-fit** are critical.

To support adaptability and optimization, both **Optimando.ai** and the broader **community** will contribute to a growing library of **highly optimized, task-specific agent templates**. These templates can be plugged into any orchestration instance, allowing users to extend or adapt the system for different industries, compliance needs, or domain-specific decision support.

The architecture is fully transparent and designed to run locally on user-owned hardware. It includes a browser extension, a lightweight desktop orchestrator, and optional device-side input apps. There is no built-in requirement for server-side logic. By default, all data remains on the user's system, and no external connections are made unless explicitly configured.

If users choose to enhance functionality by connecting to cloud-based language models, they remain in complete control over what data is shared. The system provides clear routing options that allow users to decide which data types are allowed to be sent to external services if the user decides to utilize external Llms. To support this even further, an optional privacy layer will be integrated that can help filter or mask

typical sensitive patterns, such as names or credentials. While this layer can reduce exposure, the user always decides how much to rely on it, and can disable any external calls entirely.

For advanced users or those who prefer additional security, the entire orchestration system can be run inside a local OS in a virtual machine (VM). This adds an extra boundary of isolation and helps ensure that the AI workflow is separated from the rest of the operating system. Setting up a VM takes only minutes and requires no deep technical knowledge. Ubuntu Desktop as example is free and an excellent OS for such a VM environment. When running in a VM, users can choose to handle especially sensitive tasks—such as authentication, payments, or sensitive messaging—directly on the host system instead. This separation ensures that privacy-critical processes remain fully insulated from the orchestration environment unless explicitly bridged.

Crucially, users who want to avoid cloud services altogether can embed local language models directly into the helper tab logic. Locally hosted Llms can run inside the browser without sending any data off-device. In this case the browser simply functions as connector to the local infrastructure.

Many newer PCs now include built-in AI acceleration hardware, such as neural processing units (NPUs) or dedicated inference cores. While current browser environments offer only limited access to such accelerators, the framework is designed as a forward-looking concept that anticipates their use in future local workflows. Today, companion desktop apps or native wrappers can already utilize these components for select tasks such as inference, summarization, or intent detection, provided appropriate integration via system-level APIs (e.g., DirectML, CoreML, or ONNX). In the future, deeper browser integration with local AI hardware could enable seamless, real-time performance improvements directly in the user's browser

environment. These hardware units could help facilitate lightweight, privacy-preserving automation without relying on cloud services for these critical parts, particularly when paired with local models and orchestration logic. These components can be used to speed up specific AI tasks—including local inference, redaction, summarization, or intent detection—directly on the user's device. The tab-based architecture also allows users to delegate distinct responsibilities to different tabs—for instance, using one tab to anonymize or pre-process data with a lightweight local LLM, while other tabs simultaneously leverage powerful cloud-based models, all within a controlled, user-defined workflow.

The overall design philosophy behind this framework is simple: the user stays in full control. Whether connecting cloud models, running everything locally, or blending both, the architecture adapts to individual preferences and privacy needs. It is a modular, forward-looking foundation for building intelligent, real-time workflows across devices—on the user's terms.

Unlike traditional systems that rely on a single control point, our architecture supports distributed, multi-source control. A user might speak into their phone, type on a desktop, and run a secondary application—all at once—while helper agents receive the combined or distributed context and provide intelligent support instantly.

The system runs entirely on user-controlled devices and includes:

- A browser extension for managing helper agents,
- A desktop orchestrator app, and

- Optional input apps on smartphones, AR/VR devices, or other connected input data sources.

There is no reliance on cloud infrastructure. All logic can be executed locally, ensuring maximum data privacy, low latency, and independence from proprietary platforms. All components are Source-available, fostering transparency, extensibility, and long-term scalability.

This architecture represents a flexible, forward-looking foundation for real-time AI workspace automation—positioned for use in enterprise, education, science and productivity environments where data control and cross-device intelligence are strategic priorities.

The system is particularly effective in augmenting active digital workflows—whether interacting with LLMs, filling out forms, configuring automations, or managing content. It supports **real-time prompt refinement**, **tree-of-thought reasoning**, **structured brainstorming**, and **logic-driven output suggestions**. In LLM-based scenarios, the orchestrator can detect chatbot completion events and inject improved follow-up prompts automatically using DOM manipulation—enabling seamless, supervised multi-step interactions. In other use cases, it can offer contextual next-step optimizations, alternative strategies, or compliance-aware modifications—all delivered in real time based on the user's intent and setup.

Use cases span a broad range of digital activities, including—but not limited to—automation design, form completion, business logic configuration, knowledge work, research, business communication, training, live-coaching and brainstorming. The system dynamically observes the user's intent by analyzing the visible input and output context within the active master interfaces. Additionally, users can define a persistent global context that reflects broader goals, project parameters, or organizational constraints—enabling the helper agents to align their suggestions even more precisely with the intended outcome. Improvements range from

GDPR-compliant alternatives to optimized strategies, refined decision paths, or context-aware workflow enhancements tailored to the user's objectives.

The update interval for detecting chatbot completion, capturing screenshot or stream-based inputs, and triggering follow-up events can be precisely configured by the user.

Multi-tab orchestration

- **Multiple master tabs per session**, including support for distributed setups where multiple users, external apps, or even robotic camera systems can act as master input sources
- **Session templates** for reusable configurations
- **Autonomous and manual feedback loop triggers**: The orchestration logic allows for feedback loops between any combination of master and helper agents. These loops can be triggered automatically based on predefined conditions, or manually by human-in-the-loop intervention. For instance, in a distributed setup, a team of AR device operators may continuously stream contextual data into the system. Desktop-based analysts or supervisors—acting as orchestrators—can monitor this data in real time and provide direct feedback back to the AR operators. In parallel, helper tabs can exchange insights or findings among themselves based on logic rules, further enhancing the feedback cycle. This enables the creation of semi-autonomous or fully autonomous workflows, which can be toggled on or off depending on task complexity, user preference, or regulatory constraints
- **Local-only, GDPR-compliant design possible (local LLMs only)**

- A strict separation between **logic/control (master)** and **browser-based execution (helper tabs)**
- Beside automatic user intent detection, a small chat window can be opened as an overlay to directly command specific agents or tools — either by text or via voice input.

Modern knowledge work often involves frequent switching between browser tabs and applications, resulting in cognitive overhead and productivity loss. Studies suggest users switch between digital interfaces over 1,000 times per day, often losing several hours weekly to simple reorientation.

Agentic AI systems seek to reduce this friction by acting as intelligent intermediaries across applications. Users can instruct such systems to retrieve data, automate steps, or manage multistep workflows across interfaces. Recent efforts such as OpenAI's *Operator*, DeepMind's *Project Mariner*, and Opera's *Neon* browser illustrate growing capabilities in web-based agentic interaction using large language models (LLMs).

Architectures across these projects vary—ranging from browser-integrated assistants to cloud-hosted control environments. Common capabilities include form handling, navigation, summarization, and task execution using multimodal input. While promising, deployment remains subject to broader industry challenges such as session continuity, transparency, and user-aligned control structures.

The Optimando.ai framework introduces a modular, source-available orchestration concept designed to operate within standard browser environments, optionally backed by locally hosted LLMs. It enables **real-time, context-driven optimization** using multiple **independent AI helper agents**, each operating in its own browser tab. These helper tabs may host **distinct LLMs such as the web-based versions of ChatGPT, Gemini,**

Project Mariner, Claude, Mistral, Grok, Llama or local open source Llms selected by the user based on the task's privacy, cost, or reasoning complexity.

For example, lightweight or low-cost LLMs can be used in helper tabs handling repetitive or less critical tasks and even form filling in helper tabs; advanced reasoning models can be reserved for complex, high-value workflows; and locally hosted LLMs may process sensitive or private information in fully self-contained tabs.

Input data to this system can be **multimodal**, including text, audio, screenshots, UI events, or camera streams. These triggers can originate from within the desktop browser or be activated remotely via smartphones or AR devices acting as **remote master agents**. Once toggled active, such devices send live input to a workstation. Contextual signals from any application—local, remote or on premise—can be 'looped through' to the master tab, effectively integrating them into the orchestration flow.

Users may operate directly at the orchestration workstation or remotely initiate tasks from other locations. The system supports both **autonomous execution** and **human-in-the-loop workflows**, enabling oversight, corrections, or adaptive feedback. This flexibility allows real-time augmentation, background execution, and hybrid workflows tailored to user preferences.

The overall architecture transforms the browser into a **distributed optimization surface** — a live environment where specialized AI agents process inputs contextually and reactively in real time. It emphasizes modularity, transparency, and cross-device orchestration, giving users full control over how intelligence is distributed and applied across their digital environment.

While still conceptual, the framework provides a **directionally unique and open model** for AI orchestration — enabling scalable, secure, and customizable workflows through heterogeneous agents and devices.

Contemporary AI agent tools are typically designed around a **primary agent architecture**, operating within a browser interface or cloud-based environment. For example, Google's *Project Mariner* (2025) has been described as an experimental browser assistant that allows users to interact via natural language. Based on public reports, it can autonomously navigate websites to perform actions such as purchasing tickets. More recent updates suggest that Mariner operates in a cloud-based environment, enabling concurrent task handling — though the interface remains structured around a single active agent session.

Similarly, OpenAI's *Operator* (also referred to as the "Computer-Using Agent") utilizes GPT-4o with vision to interpret and interact with web pages. Operator appears to manage multiple tasks by launching separate threads or sessions, but each instance represents a distinct agent working within its own conversational context. Browsers themselves are beginning to integrate AI agents. Opera Neon (2025) is billed as the first "agentic browser": it embeds a native AI that can chat with the user and a separate "Browser Operator" that automates web tasks (forms, shopping, etc.). Opera also demonstrates a more ambitious "AI engine" that, in the cloud, can work on user-specified projects offline and do multitasking in parallel. However, Opera's agents are proprietary, deeply integrated into one browser, and not open for user modification. Opera One (a related product) has introduced AI Tab Commands: a feature where a built-in assistant can group or close tabs on command (e.g. "group all tabs about ancient Rome"). This helps manage tab clutter, but it still uses a single AI interface per browser to organize tabs, without supporting multiple cooperating agents.

**Input Data /
Screenshots**



AR Glasses

**Helper Tabs for
Contextual Realtime
Optimization
with AI**

Outside the browser domain, research on LLM-based Multi-Agent Systems (MAS) is rapidly growing. Tran et al. (2025) survey LLM-MAS and note that groups of LLM-based agents can “coordinate and solve complex tasks collectively at scale”. Emerging orchestration frameworks (e.g. AWS Bedrock’s multi-agent service or Microsoft’s AI Foundry) allow specialized agents to collaborate under a supervisor, and enterprises are experimenting with central “Agent OS” platforms that integrate many agents. But these systems operate at the level of backend services or applications, not at the level of coordinating a user’s browser environment. Crucially, we found no published work on orchestrating multiple AI agents distributed across browser tabs as a unified workspace.

Privacy and Control. As AI agents increasingly interact with web interfaces and user data, privacy and control have become central design concerns. Existing projects such as Opera’s *Neon* emphasize that automation runs locally to preserve users’ privacy and security. Similarly, OpenAI’s *Operator* allows users to manually intervene in sensitive interactions via a “takeover mode” and is designed to avoid capturing private content without user intent.

The framework developed by Optimando.ai adopts a similar privacy-first philosophy. All orchestration components are **self-hosted** and run within the user-controlled environment. **No data is transmitted to any [wrcode.org server](#).** The coordination logic and agent communication infrastructure are source-available and designed to operate under user ownership and configuration. Users may choose to deploy the framework on local machines, private servers, or trusted edge devices depending on their needs.

While remote input streams (e.g., from smartphones or AR devices) may transmit data over the internet to the orchestrator, all transmission paths and endpoints are defined and controlled by the user. Optimando.ai

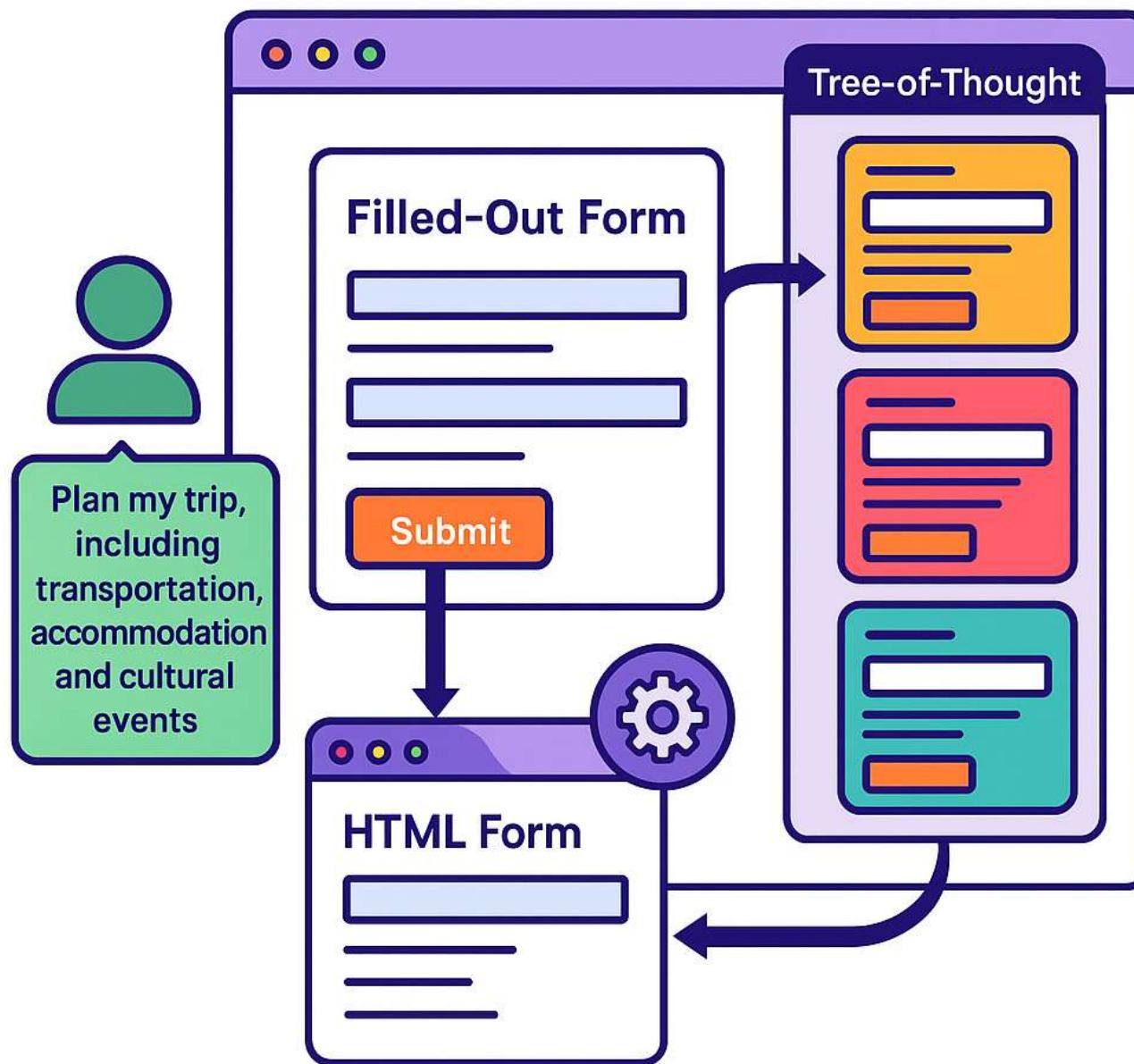
does not operate or provide any backend services that receive, log, or process this data. The system's observation is also strictly limited to in-browser content in explicitly configured tabs. There is **no tracking of desktop activity or full-device behavior**.

Browser helper tabs don't host agents; they **render outputs**. Reasoning results from one or more AI agents are executed by the orchestrator (via API or local runtime) and routed into **configurable display grids** across one or more browser tabs for side-by-side viewing, using either a locally hosted LLM interface or a third-party web-based LLM (e.g., ChatGPT, Claude, Gemini). The **choice of LLM, its operational mode (cloud or local), and any associated data sharing are entirely the responsibility of the user**. Optimando.ai has no control over the behavior, data retention, or processing methods of any third-party services the user may connect.

Proactive DOM Manipulation via Helper Tabs with Tree-of-Thought Variant Expansion and Detached UI Controls

The *optimando.ai* framework introduces a browser-centric, tab-based agent orchestration model in which helper agents operate in isolated execution environments. These agents proactively analyze structured context from the user's active session, manipulate DOM elements, and generate fully rendered outputs — all displayed outside the main interaction tab in a clean, detached format.

Unlike traditional single-path AI tools, this architecture supports **multi-path reasoning** and **alternative pre-filled results** (even before the user starts to begin with filling out fields in the master tab), activated only when the user explicitly requests exploration via a Tree-of-Thought mechanism or other AI generated options that are displayed in the display slots.



 **DOM Execution in Background Helper Agents**

Each helper Agent can act as an autonomous agent capable of:

- Parsing DOM structures from contextual input (e.g., a form, booking flow, legal interface).
- Proactively filling out entire forms or interfaces in its own environment — without affecting the active user tab.
- Executing reasoning, retrieval, and content augmentation based on the user's prompt and global session state.
- Generating a **complete, actionable version** of the task (e.g. a filled form, contract draft, booking process) and sending it to the **orchestrator for display**.

 **Generalized Example: Cross-Platform Action Planning**

A user enters a request via the master input interface:

"Please plan my trip, including transportation, accommodation, and cultural events."

This triggers a coordinated, multi-agent response:

- A helper agent decomposes the task and distributes subtasks across several pre-defined services (e.g., airline booking portals, hotel sites, ticket platforms).
- Security-first logic ensures that no untrusted website is opened without the user's explicit domain whitelist. If broader exploration is needed, sandboxed environments like Google Mariner may be used — though this introduces profiling considerations.
- User interaction is required for all sensitive operations. The system will not perform actions on its own. Tabs must be opened by the user (e.g., to authenticate into Gmail or a booking portal), or previously opened, authenticated sessions are reused.

Once results are gathered, the best-matching itinerary is synthesized and displayed visually across dedicated output slots — rather than a single frame. This leverages the orchestration system's adaptive layout, which utilizes available screen space intelligently, in contrast to conventional systems like Google Mariner that condense content into one unified panel.

Each itinerary component is annotated with metadata (e.g., pricing, provider rating, availability).

The master input tab can be toggled off or minimized to reduce distraction. Users may then focus entirely on the results and interact with the system directly via the display slots — including continuing or expanding the session using Tree-of-Thought expansion.

○ Interactive Output Features

The top result includes functional, user-controlled elements — which may be customized or even generated contextually by AI — but all actions require explicit user approval.

- Submit – Fully functional button, but only executes upon explicit user interaction. The logic behind this can be customized or extended.
- Tree-of-Thought – Reveals pre-processed reasoning paths and diverse alternatives:
 - Different platforms or service providers
 - Alternative travel schedules
 - Price tiers or bundled offers
- Compare & Combine – Allows hybrid planning (e.g., mix Flight A with Hotel B)
- Explain – Displays the reasoning and decision metrics used to generate the top result
- Augment – Enables real-time refinement (e.g., "add airport transfer", "include insurance")

The orchestration system shows only the primary recommendation by default. Users can dive deeper via expansion tools, ensuring an uncluttered yet powerful interface. This configuration empowers

users with intelligent, modular autonomy — decisions are enhanced by AI, but always confirmed by the human.

Additional Example: Structured Form Completion

A user begins interacting with a financial or tax-related form.

- A helper tab interprets the structure and populates it with relevant user data (pre-uploaded context data), rule-based logic, or AI-completed content.
- The best-matching version (e.g., a conservative filing strategy) is displayed first.
- If ambiguity or multiple valid interpretations are detected, a **Tree-of-Thought button** becomes available:
 - When clicked, it reveals other fully generated versions (e.g. aggressive vs. conservative deductions, business vs. private allocations).
 - Each is independently rendered, side-by-side or in cascading slots, ready for user review and approval.

What Sets This System Apart

Most existing AI assistant systems:

- Operate in a single DOM/UI context

- Most conventional AI assistants do support multiple outputs, but these are often rendered inline within the same interface, making structured comparison difficult. The suggestions are usually presented in a linear or dropdown format, without architectural separation or agent-driven execution. This makes deeper exploration—such as Tree-of-Thought reasoning or cross-platform branching—difficult to manage or scale effectively.
- Do not support parallel reasoning paths or structured exploration

By contrast, the *optimando.ai* framework:

- **Isolates execution from interaction** — reducing risk, clutter, and interference
- **Supports cross-platform reasoning and multi-source orchestration**
- **Generates and renders variants only on demand**, using a **Tree-of-Thought button**
- Encourages structured, user-controlled decision-making, rather than opaque automation

Architectural Benefits

- **Non-invasive assistance:** The master interface remains unchanged; all suggestions appear in dedicated display zones.
- **Session-safe integration:** Helper tabs inherit authentication from the user's active browser session.

- **Parallel reasoning at scale:** Each agent tab can target a different platform or strategy — executed in true parallelism.
- **Human-in-the-loop control:** Results are passive unless approved; the user always decides what to apply or explore further by default.

Integration of Metaverse Interfaces into the Browser-Based Orchestration Framework— Future Outlook (Optional)

While the orchestration framework is based on browser tabs, AI agents, and configurable templates, the **WR CODE** architecture is inherently extensible to virtual environments—**without requiring deep integration into game engines or metaverse platforms**. In this extended use case, the orchestration continues to run on a conventional computer, with AI agents distributed across browser tabs as defined by configuration files. Certain helper agents—originally designed to operate in the background—can optionally be linked to **visual representations within a 3D environment**, such as NPCs (non-playable characters). These NPCs serve purely as **front-end proxies**; the underlying logic, memory, and decision-making processes remain in the external orchestration system.

For instance, in a virtual shop scenario, a user may interact with digital products and approach a cashier NPC to initiate checkout. The NPC itself does not contain embedded logic; rather, it connects to a designated browser tab acting as a helper agent. This tab interfaces with external systems—such as a shop backend, support knowledge base, or legal compliance service—via automation platforms like **n8n** or **MCP-connected agents**. The orchestration layer interprets the user's intent (e.g. purchasing specific items) and generates

structured outputs, such as a purchase summary, legal disclaimers, and pricing details. These are presented in-world via spatial overlays or embedded screens—**visual equivalents of the system's display slots**. The user may confirm or cancel the transaction directly within the metaverse, triggering corresponding real-world actions through the connected orchestration backend.

Beyond service and commerce scenarios, this architecture can also be extended to **orchestrate real-time AI-controlled NPC teams**. In such cases, a user (e.g. a player, moderator, or team leader) can issue instructions that are routed through the orchestration backend, which interprets intent and assigns tasks to corresponding NPC agents based on preconfigured logic. This enables the coordination of **multi-agent NPC behaviors**—for example, managing logistics crews, training groups, or support units in real time—without requiring native AI infrastructure within the 3D environment itself.

This decoupled architecture allows metaverse applications to benefit from **advanced AI orchestration, decision logic, and automation**—while keeping the virtual environment lightweight and modular. By separating interaction from execution, it enables fast, maintainable integration of intelligent workflows into immersive spaces, using the same **tab-based orchestration layer** originally developed for browser-native contexts.

⌚ Autonomous NFTs (aNFTs) — Future Outlook (Optional)

As an optional future direction, Autonomous NFTs (aNFTs) could complement WR CODE's orchestration capabilities by enabling new ways to bridge real-world events with decentralized digital assets and automations:

- **Policy Reference:** NFTs could reference global or local policies to guide AI-driven decisions and trigger actions based on predefined conditions.
- **Proof-of-Action:** Real-world events (e.g., QR scans, geofence triggers) could mint or update NFTs as verifiable records, optionally anchored on decentralized systems such as IOTA.
- **Autonomous Rewards:** NFTs could hold assets, rights, or permissions that are dynamically unlocked through verified actions, providing tamper-proof, decentralized incentives.

To enhance privacy, proofs of action or ownership could be anchored without revealing identities. NFTs themselves could also be hashed and used in zero-knowledge proof (ZKP) systems, allowing users to demonstrate NFT ownership or eligibility without exposing wallet addresses or transaction details.

Importantly, **WR CODE's** AI-driven automation templates are hashed and anchored on Solana to ensure trustless verification of AI automation flows. This would allow third parties to independently verify that specific, immutable rules triggered certain automations—enabling AI-driven processes that are both transparent and verifiable without centralized trust. In addition to the templates, also the executions of AI automations themselves could be hashed and logged, providing an immutable, verifiable chain of evidence that documents what rule triggered what action at what time. This ensures that both the existence and the consistent application of rules can be externally validated. In addition, **WR CODE** could serve as a backend orchestration layer for metaverse platforms or digital identity systems where NFTs or AI-controlled NPCs play a central role. By combining real-world triggers (WR Codes, geofencing) with

virtual world actions (NFT updates, digital rewards, AI-driven responses)—all optionally anchored on decentralized systems—WR CODE could enable seamless, privacy-respecting automation that bridges physical and virtual environments. These ideas illustrate potential future extensions beyond WR CODE's core innovation and would require further research and development to ensure security, privacy, and decentralized enforceability.

Privacy by Architecture: Local Control Through Tab-Based Orchestration

Modern AI automation presents a paradox: its usefulness grows with access to user context — but so do the privacy risks. *optimando.ai is designed to support users in protecting their privacy through a layered, modular system architecture. While it cannot prevent all risks — especially when users voluntarily share personal data with external services — its structure enables masked automation, local execution, and transparent control over agent behavior.*

As a source-available platform without vendor lock-in, optimando.ai invites continuous improvement and innovation. Its flexible design allows the community to build on a privacy-aware foundation that evolves with real-world needs.

At the heart of the system is a browser-centric tab orchestration model, which allows users to automate workflows while preserving control over how, when, and what kind of data is processed.

 Layered System Components for Structural Privacy

This is not abstract or theoretical privacy; it is embedded directly in the system's architecture and enforced through clearly separated roles and control layers.

| Component | Privacy Role |
|-------------|---|
| | <p>The user's primary interaction layer — which may include but is not limited to browsing, shopping portals, SaaS tools, embedded apps, or even external visual inputs (e.g., camera feeds, video streams, robots). This is the starting point for all activity.</p> |
| Master Tab | <p>Because the Master Tab handles real-time input directly from the user and external sources, <i>optimando.ai</i> cannot intercept or mask visual or textual content before it reaches cloud-connected systems. Users must be aware that any personal data or visual context shared here can be exposed, especially if routed to external AI services.</p> <p><i>optimando.ai</i> provides tooling to structure, automate, and augment downstream tasks — but <i>*cannot enforce privacy at the source input level. The user remains in control.</i></p> |
| Helper Tabs | <p>Sandboxed environments where AI agents operate. These tabs are designed to receive masked, preprocessed, or synthetic data, especially when handling structured automation tasks like form filling or contextual suggestion.</p> |

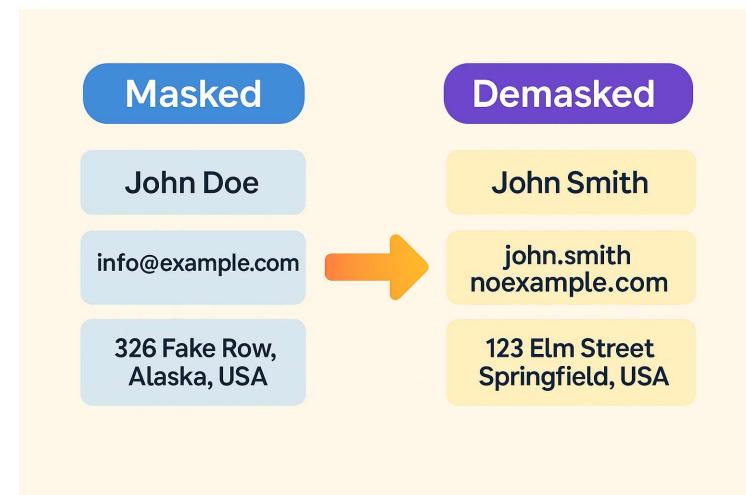
| Component | Privacy Role |
|------------------------------|---|
| | However, the degree of masking depends on user configuration and workflow context. <i>optimando.ai</i> provides mechanisms to prevent the exposure of raw personal data, but cannot enforce this universally. It is the user's responsibility to ensure that privacy settings are correctly applied and that no sensitive inputs are embedded into prompts or data sent to cloud-based models without proper masking. |
| Input/Output Coordinator Tab | Routes logic, manages masking/demasking workflows, and controls agent output. It ensures context flow is constrained and reversible locally. |
| Display Slots | Passive render areas. They display results from helper agents but do not trigger outbound data flow. If the user chooses to submit a form, it's sent directly to the target website, not to any AI system. |

This orchestration strategy allows privacy protection at the execution level, not just via policy.

🛡️ What's Protected: Contextual Data Masking

The system applies local masking and demasking to sensitive fields before sending them to helper agents. Supported transformations include:

- Names → pseudonyms (e.g., “John Doe”)
- Addresses → synthetic or shifted variants
- Dates → randomized or offset
- Numbers → obfuscated mathematically (e.g., -30% for income)



Example: Privacy-Preserving Form Automation Using Local LLMs and Embedded Context Memory

When a user visits a structured form — such as a job application, tax return, or official registration — the optimando.ai orchestration framework can detect this intent and assist with intelligent, local form filling.

If the feature is enabled, the system loads the same page in a helper tab, where it attempts to prefill the form automatically. It does so using relevant user data (e.g., resumes, documents, income reports) that are stored locally — optionally within an encrypted container (e.g., VeraCrypt). These files are simply stored on the user's device.

A local LLM (e.g. Mistral) — optionally within an encrypted container (e.g., VeraCrypt), parses the content and embeds it into a private vector database (e.g., Qdrant). From this, it can derive context and field-level data (names, skills, income, etc.) and use DOM manipulation to fill in the form.

The resulting form is shown in a display slot — for example, on a second screen — allowing the user to check, adjust, and submit manually.

Even more broadly, if a user watches a YouTube video (master tab)—for example, a product walkthrough of a new AI tool—slave agents can assist by cross-referencing this information with the user's current goals or projects. One agent might highlight how the showcased tool could be integrated into the user's existing tech stack. Another might suggest more efficient or better-suited alternatives based on predefined system constraints or preferences. A third agent could retrieve relevant use cases or success stories, helping the user assess practical value. Together, they act as a live research and recommendation engine—turning passive content consumption into actionable insights aligned with the user's broader objectives.

The user sees outputs in display slot even from possible other remotely interconnected helper tabs or input sources as suggestions or context inserts in real time. This creates an optimization loop: the user steers the main task, customizable helper tabs continuously augment it automatically, and the user approves or refines the results. The human stays “in the loop” at every step, aligning with best practices in trustworthy AI.



 Local by Default — But Flexible and User-Controlled

The architecture is designed around privacy-first principles, but remains adaptable:

- By default, all tasks involving personal data (PII) are handled exclusively by the local LLM.
- Users retain full control — they decide whether and when to allow external assistance.
- The local model performs all tasks it can handle, especially any involving sensitive data.

This privacy-by-design approach makes it possible to build automation capabilities for white-labeled websites similar to those seen in emerging agentic browser systems — such as Google Project Mariner or Manus — but without exposing profiling-relevant or PII data to third-party services.

 Optional Cloud Reasoning — With Controlled Isolation

If the local model cannot solve a more complex reasoning task (e.g. legal interpretation, tax optimization, logic chaining), the user may choose to enable external LLM assistance. In these cases:

- Only non-PII and abstracted fragments are sent to cloud LLMs.
- PII and sensitive identifiers stay on the local machine.
- Profiling-resistant techniques are applied, such as:

- Task splitting across providers
- Field-level masking/demasking
- Tree-of-Thought confusion, generating variants to conceal true intent or values

Additionally, when users directly type information into the master tab (instead of using helper workflows), the system may trigger additional safeguards like DOM-based submit delays or selective field protection.

Strategic Privacy Design — Without Overpromising

Optimando.ai is a source-available, evolving framework. Privacy protection is a core design goal — but users should be aware:

- Not all features are active from day one.
- It is built to grow with community input, legal requirements, and innovation.
- While it applies state-of-the-art techniques to reduce data exposure and profiling risk, no system can guarantee absolute protection.
- The user stays in control, and bears responsibility for how the system is configured and used.

Unlike many SaaS automation platforms, optimando.ai never assumes full ownership of user data. The goal is to enable powerful AI-assisted workflows — such as "autopilot" for complex digital tasks — without the user needing to hand over sensitive information.

 **Optional Encryption – Recommended Best Practice**

While WR CODE does not enforce encryption itself, users can and should take steps to protect their sensitive data. This includes:

- Encrypting files containing personal documents (e.g. resumes, tax PDFs)
- Storing local LLM model files and vector databases inside encrypted containers (e.g. via VeraCrypt or similar tools)
- Ensuring that any local embeddings involving PII or data that can be used for profiling are also secured at rest
- WRVault for secure data storage

Since embedded vectors or fine-tuned LLM memory can contain sensitive content, they should be treated as equivalent to raw data files from a privacy standpoint.

Encryption is entirely optional, but it is strongly recommended — especially for users working on shared machines, mobile devices, or in regulated environments.

This best-practice approach allows users to maintain complete control over their local data environment, without depending on external providers or complex infrastructure.

Local-Only Form Submission

When an agent completes a form, it appears in a Display Slot. If the user chooses to submit:

- The submission goes directly to the target site (e.g., a booking or government portal)
- No AI model or external agent sees the unmasked data
- Execution happens via the user's own session and browser state

This makes the entire interaction local, transparent, and user-driven.

What Is the Obfuscation Layer?

The Obfuscation Layer is an optional privacy-first component within WR CODE's orchestration system. It's designed to prevent oversharing with LLMs, especially when cloud-based APIs are involved (e.g., OpenAI, Anthropic, Gemini, Deepseek, Grok).

Once enabled, this layer actively monitors outgoing prompts and applies a mix of filtering, masking, prompt splitting, obfuscation, and routing logic. Its goal: to make AI helpful—but ignorant of your private life and strategic context.

How It Works: Core Tactics

1. Prompt Classification & Criticality Detection

- Prompts are broken down into logical units.

- Each unit is scored for risk (e.g., names, dates, salaries, roles).
- Critical content is routed differently than harmless filler.

2. Hybrid Prompt Splitting (Local + Cloud)

- Critical fragments are routed to locally hosted LLMs (e.g., LLaMA 3, Mistral).
- Non-sensitive content is routed to cloud-based LLMs.
- This allows performance without full data exposure.

3. Masking & On-Demand Demasking

- Personally identifiable information (PII), sensitive numbers, or unique metadata are masked locally.
- Only upon user confirmation can these be used in output.

4. Prompt Obfuscation & Confusion

- Fake sub-prompts and decoys are generated using Tree-of-Thought Confusion.
- The goal is to break intent recognition by introducing false paths.

5. User Nudging & Input Delay

- If sensitive data is detected in raw input, the system pauses and asks for confirmation.

- This prevents accidental leaks.
-

A Real-World Example: The Travel Booking Trap

Imagine Lara, a freelance journalist working on a sensitive investigation.

Using a cloud-based LLM via WR CODE, she:

- Books a Flight
- Books a hotel
- Rents a car
- Reserves a table at a restaurant
- Fills out insurance forms
- Writes a press letter

In doing so, she enters:

- Name, birthday, passport details
- Locations, dates, and affiliations

- Purpose of visit and logistics

If this data is sent raw to a cloud LLM, it could be:

- Logged permanently
- Used to train future models
- Subpoenaed in foreign jurisdictions
- Leaked during breaches

For Lara, this is more than a privacy concern—it's a risk to her safety.

Ongoing Research, Evolving Defenses

The Obfuscation Layer is a live research field. It continues to evolve with:

- Live sensitivity scoring
- Role-based redaction profiles
- Math over encrypted values
- Tree-of-Thought decoys
- ZK-proof-based masking

- A single prompt is fragmented into reasoning units and distributed across multiple cloud-based AI models to prevent full visibility at any single endpoint.

The goal is clear:

To reduce the LLM's ability to extract meaning while still letting it perform the task. It starts experimental but will evolve over time into a reliable obfuscation layer.



Tree-of-Thought Expansion & Intent Obfuscation

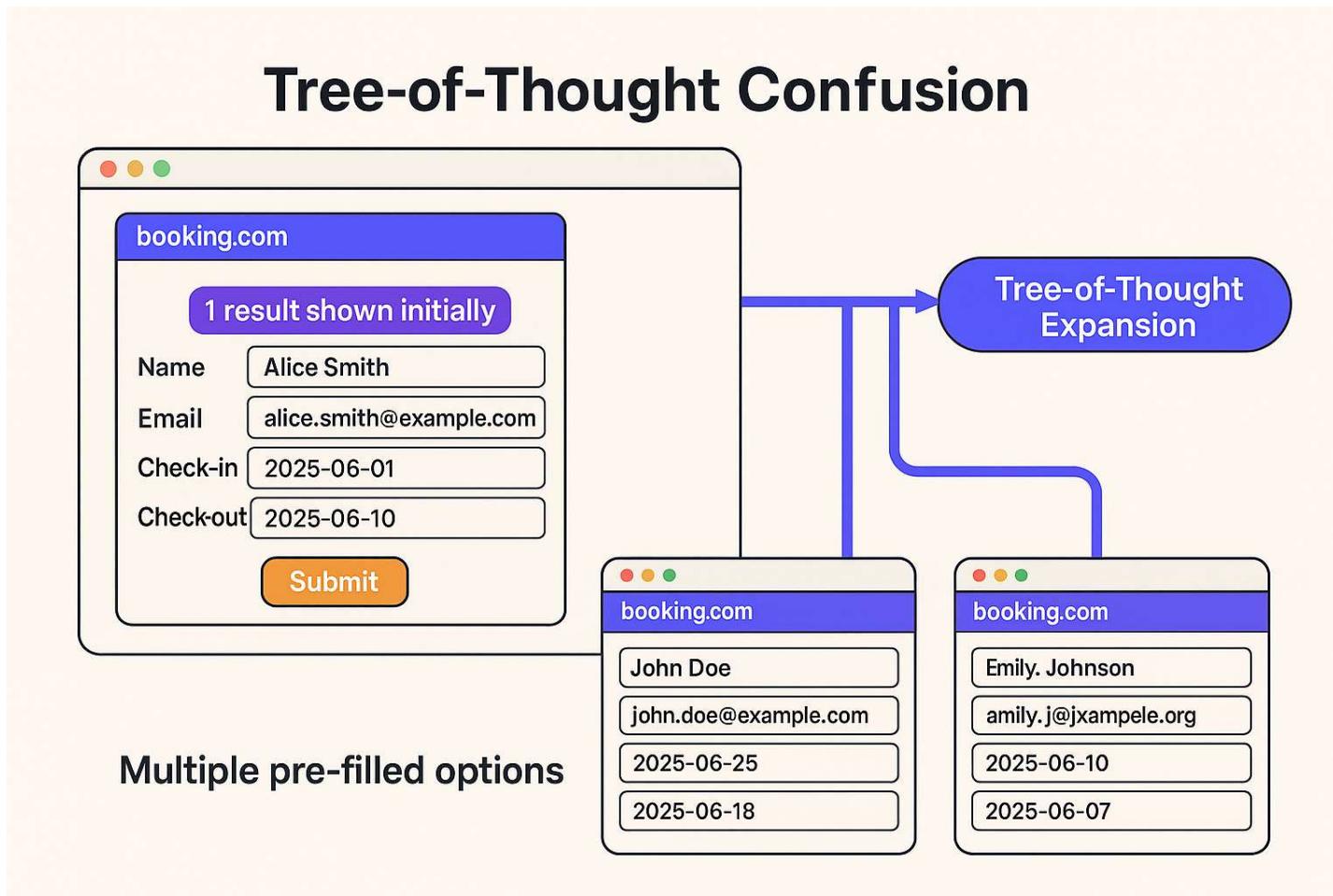
Thanks to its distributed helper-tab design, *optimando.ai* enables a novel privacy feature: intent obfuscation through variant generation.

- Multiple helper agents generate alternative form completions (e.g., booking dates, tax strategies, application paths).
- Only some are “real” — but the AI doesn’t know which. The user sees only real results.
- The others act as plausible decoys and are processed in the helper tabs.

This makes it difficult for even a remote AI system to infer true user preferences or intent — especially in planning and decision-heavy workflows. The concept is simple: the more plausible paths exist, the harder it is to profile the user. And it’s not just a theoretical concern. History shows us that even the most trusted cloud providers and corporations have suffered massive data breaches. Customer records, internal tools, and sensitive documents have been leaked—and sold in darknet markets.

Organized cybercrime groups are technically advanced, often better funded than security teams. Some users use any chatbot suggestion they find trending online—without hesitation, without context, without even having the slightest concerns. But what happens when these “trusted” platforms are infiltrated, backdoored, or misused by insiders? In some parts of the world, even governments are actively complicit in criminal operations—spying on dissidents, persecuting whistleblowers, or exploiting data from compromised platforms. We live in a time of war, bloodshed, and hatred in the streets—where digital trust

is fragile and information can be weaponized. For high-risk individuals, casual interaction with AI systems is not just risky—it can be dangerous. It's not all sunshine out there. The Obfuscation Layer is designed for those who are most vulnerable to these threats.



Conceptual Feature: Tree-of-Thought Confusion and Profiling Risk Mitigation

A privacy-first feature currently in internal design is the Tree-of-Thought Confusion mechanism, combined with real-time profiling risk detection and dynamic masking/demasking overlays in the Master Tab. It is designed to mitigate unintended profiling when users interact with LLM systems in sensitive contexts such as healthcare, legal, or financial domains.

Example Use Case: "Please find a list of oncologists near Berlin for my ongoing cancer treatment."

This seemingly routine request reveals highly sensitive personal data. With the growing use of LLMs — across both trusted and opaque platforms — many users remain unaware of the long-term profiling risks. If such profiles are logged, shared, or leaked, the consequences could be significant — from discrimination to surveillance.

This concern is particularly relevant for:

- **Journalists and whistleblowers**, researching or reporting sensitive topics
- **NGO workers or activists**, operating in authoritarian or high-risk environments
- **Professionals and individuals** handling confidential medical, legal, or financial data

For these groups, accidental exposure of intent, interests, or vulnerabilities through AI-driven interfaces poses a real and growing threat.

Technical Outlook: Local AI and Real-Time Protection

The proposed system envisions a local privacy layer powered by lightweight LLMs such as Mistral 7B, which already run on most modern workstations (32–64 GB RAM). These models could enable:

- On-device detection of sensitive input and real-time overlay augmentation
- Dynamic obfuscation and decoy logic (Tree-of-Thought style)
- Intelligent result routing and masking within the orchestration layer

By avoiding cloud dependency, the system offers privacy-preserving workflows tailored for security-conscious users.

Although such augmentation is technically feasible, reliable implementation—especially DOM manipulation, prompt interception, and UI-level delays—is non-trivial. Many modern web apps use dynamic frontends (e.g., React), requiring resilient engineering to ensure consistency across sessions and platforms. For this reason, these capabilities are not part of the initial release, but are being explored for future development.

Systems with limited resources may fall back to simpler rule-based methods, such as keyword filters or DOM-based context tagging, which still offer value in less complex environments.

Always keep in mind

This system mitigates risks, but it's not magic.

- If a user types their real name, address, or salary into a public chatbot — that data is exposed.
- If a task requires semantic accuracy (e.g., legal document interpretation), masking may break the logic.

optimando.ai does not claim to prevent all leakage. It offers technical control where feasible, and leaves informed decisions to the user.

The User Is Always in Control

Privacy in *optimando.ai* isn't based on trust — it's based on architecture. But it also assumes that:

- The user understands what happens where
- The user decides what is masked, unmasked, or revealed

The system never processes or acts autonomously on private data without explicit user permission.

All integrations, logic flows, and connected services are defined by the user. The user remains fully in control of which tools are used, how data is routed, and when execution is allowed. This is human-in-the-loop AI orchestration — not blind automation.

 **For more control: API-Based Chatbot Mimicry**

To overcome the limitations of DOM manipulation in dynamic web apps, the system can mimic chatbot interfaces using a controlled, API-connected frontend. This enables:

 **Prompt Filtering**

Apply real-time sensitivity checks before sending prompts to the model.

 **Decoy Injection**

Generate and route plausible decoy prompts to obfuscate user intent.

 **Overlay Augmentation**

Add dynamic masking/demasking layers directly in the controlled UI.

 **Output Separation**

Ensure only the real response is shown in the display slot; decoys remain background-only.

 **Built-In Delivery**

Since the orchestration tool includes a locally hosted web interface, this functionality is available out of the box for users who need more control.

 **Note:** The mimicked API interface may not replicate all features of the provider's original web UI.

Some elements like UI-specific memory, live suggestions, or threaded context may differ, depending on how the provider structures their API.

Summary: Real Privacy, Built In

optimando.ai enables advanced AI automation — form completion, variant suggestion, workflow support — without exposing user identity, sensitive values, or decision logic to remote models. Through tab orchestration, masking, and local submission, the system offers:

- Agent-level PII protection
- Tree-of-Thought obfuscation
- Local demasking and rendering
- Full user control at every step

What makes this system stand out is not just what it does — but what it lets the user choose not to do.

On-Premise Augmentation Overlay for Masked Reasoning and Guided Interaction

Purpose: The Augmentation Overlay is a local-only feature that improves interpretability and control in sensitive or complex software environments. It works in two complementary modes:

- Form Field Cloning & Conversational Autofill – input fields are cloned into a transparent overlay where the user can *talk to the system* to guide suggestions, variations, augmentation and placement.
- Navigation Overlay & Interactive Guidance – live interfaces are augmented with markers and tooltips that respond to natural language queries, helping users navigate complex workflows.

Both modes execute entirely on-premise, ensuring no sensitive data leaves the workstation.

Core Concepts

Form Field Cloning & Conversational Augmentation

- Input fields are automatically detected and cloned into a transparent, draggable overlay. (auto-mode)
- The user can select a screen area, which is then automatically augmented—or, if form fields are detected, cloned onto the overlay.

The overlay adapts via:

- Adaptive Transparency – opacity shifts dynamically so underlying content stays readable.
- Quick-Switch Minimization – collapse to a small icon or toolbar, restore instantly with one click.
- Conversational Interaction:
- User can *talk to the overlay* (voice or text):
 - “Where do I find the missing tax-id number?” (Local WR Vault will be gradually filled with data)
 - “I have an UG registered as company form.” (Filling and augmentation will be adopted accordingly)
 - “This data here is not correct. The billing year was 2024. What do I need to change?”

- Overlay responds by adapting augmentation — showing WRVault suggestions, generating LLM variations, or adjusting positioning.
- Suggestions are provenance-labeled (Vault / Model / Web / Synthetic).
- A variation icon allows the user to generate multiple alternatives and shuffle through them via slider.

Navigation Overlay & Guided Interaction

- Instead of cloning, the overlay augments the existing interface with contextual markers.
- Users can ask questions such as:
- “*Where do I find the API key?*”
- “*Which menu lets me configure OAuth?*”

Overlay responds with:

- Highlights on relevant buttons or menus
- Arrows showing navigation paths
- Tooltips with contextual explanations
- Adaptive Transparency ensures markers never obstruct content.
- Quick-Switch Minimization lets the overlay collapse instantly into a small icon.

Human-in-the-Loop Validation

- In form mode: users explicitly confirm which suggested values to accept.
- In navigation mode: users validate whether the highlighted path is correct.

Conversational inputs keep the loop interactive while preserving user control.

Example Use Cases

- **Conversational Form Autofill:**
User: Selects screen area for tax form filling → overlay duplicate pulls secure values from WRVault, offers alternatives, augments the fields automatically. (AI detected user wants to fill out form User: "No I meant 2024" -> Augmentation and pre-filled data adapts on the fly. User: Clicks a small icon "Apply all fields")
- **Scenario Exploration:**
User: "*Give me three possible invoice descriptions.*" → overlay generates and shuffles variations, user selects preferred option.
- **Navigation Assistance:**
User: "*Where do I set up notifications?*" → overlay highlights Settings → Notifications, adapts transparency to keep labels visible.
- **Quick Workflow Control:**
User: "*Hide the overlay until I'm done with this section.*" → overlay minimizes, leaving a floating icon.

Security Principles

- **Local-Only Execution** — no sensitive data leaves the workstation.
- **Conversational but Controlled** — overlay adapts to voice/text commands, but never autofills or navigates without user confirmation.
- **Transparency & Provenance** — all suggestions are labeled with source origin.
- **Fail-Closed Defaults** — uncertain cases revert to manual input.
- **Adaptive & Minimizable** — overlay is always available but never intrusive.

Outlook: The **Augmented Overlay**, is an extension of the WR CODE Optimization Layer, manifests in practice as an **UI Optimization Overlay**: a transparent, adaptive system that merges data augmentation and interactive guidance into a single user experience. By supporting conversational input, adaptive and command-driven, context-responsive UI optimization, the overlay becomes both a reasoning observatory and a real-time optimization layer for complex backend automation workflows. Sensitive data remains local by default, while critical steps such as inserting personal information or triggering workflows can be bound to password confirmation or other safeguards. This human-in-the-loop approach ensures trust, transparency, and full user control.

The **Augmented Overlay** applies the principle of real-time optimization directly into existing user interfaces without altering them. Standard augmentations—such as inline calculations, diagram creation, or automated email drafting—are executed locally through fine-tuned LLMs for speed,

while more demanding tasks can be delegated to the cloud if required. Every interface can be tailored: users may choose from built-in options or define new overlay icons that link to backend automations (e.g. via n8n or MCP). These augmentations are triggered by DOM listeners, pattern recognition, or similarity detection using OCR on UI screenshots and descriptions. For example, the system can place a “send email” icon next to any detected address or overlay a live chart on a financial dashboard in response to a single voice command, a text instruction, or a pre-defined pattern match.

Augmented Overlay Widgets and Structured Display Grid Allocation

Beyond automatic client-side augmentation, the system defines an optional **WRCode Augmented Overlay Widget** that website owners can embed directly into their pages. This widget allows websites to expose dedicated regions—ranging from individual fields to entire **display grid sections**—that the user’s orchestrator can attach to. In these allocated areas, users can configure their own AI instructions, real-time analyses, or automated workflows powered entirely by their local WRCode orchestrator. While the augmented overlay can autonomously enhance interface elements, native widget support enables website owners to explicitly mark larger content blocks or multi-column sections for structured AI interaction, such as comprehensive product analysis, stock research dashboards, comparison matrices, form guidance, or domain-specific automation panels.

All execution remains local on the user’s device: the website never processes the AI prompts, outputs, or context data. By supporting this optional standard, website owners can offer richer, predictable, and privacy-preserving AI functionality—enabling visitors to integrate the site

seamlessly into their personal automation environment while preserving full user control and backend independence.

By combining local-first execution with optional cloud augmentation, the **Augmented Overlay** transforms multi-minute routines into instant, explainable actions. At the same time, it remains fully customizable to individual workflows, providing a consistent framework for secure, adaptive, and efficient interaction. In this way, it not only accelerates daily tasks but also lays the foundation for the longer-term **WR CODE** vision: a self-optimizing environment where orchestration flows, tools, and interface augmentations can eventually be autogenerated and continuously refined in real-time.

The **Augmented Overlay** aims to anchor underlying elements so they remain aligned during scrolling or window movements, while leveraging cursor-tracking, gesture patterns, and tool-dependent cursor designs to enhance text/voice communication and dynamically adapt its behavior to the user's context — for example, a lasso-style cursor movement combined with the question "*Is this the field where I need to insert the text?*" allows the overlay to clearly understand the user's intent.

The Augmented Overlay is a subsystem of the WR-Code standard and works on any website or software product by default. However, publishers can optionally register their product and embed a WR-Code that provides structured Augmented Overlay context data (e.g., UI element descriptions, menu hierarchy, navigation steps). When such a WR-Code is detected, the Orchestrator verifies it and automatically retrieves the corresponding context dataset. This enables the Augmented Overlay

to guide users with much higher accuracy, improving usability with minimal publisher effort—while the system still remains fully functional, though less precise, for non-registered websites and software.

A Template-Driven, Component-Verified Architecture for Secure Real-Time Automation

Mini Apps, Workflows, AI Agents, and Augmented Overlays Through Deterministic Composition

This work presents a secure, template-driven architecture for constructing Mini Apps, Workflows, AI Agents, and Augmented Overlays in real time.

In contrast to systems that generate and execute code at runtime, this architecture performs deterministic composition from cryptographically verified components. This yields predictable behavior, compatibility with browser security mechanisms (CSP, Trusted Types, sandboxed execution), and strong integrity enforcement.

All dynamic behavior originates from plain-text templates that describe semantic intent, structure, and constraints, but never embed executable code. The orchestrator interprets these templates, matches them against secure component libraries, and instantiates the resulting behavior locally on the user’s device.

Where capabilities are missing, the system may—subject to explicit user consent—submit structured component requests to wrcode.org, enabling controlled evolution of the component libraries.

1. Plain-Text Templates as Universal Behavioral Descriptors

All dynamic entities—Augmented Overlays, Mini Apps, Workflows, and AI Agents—are expressed as uniform plain-text templates. These templates describe:

- Semantic targets: UI elements, data sources, workflow states
- Intended operations: analysis, transformation, verification, guidance
- Presentation requirements: overlay forms, summaries, contextual indicators
- Interaction and privacy constraints: consent requirements, visibility, masking
- Triggers and control flow: when and how behavior may be activated

Templates are authorable by publishers and users, and may be refined or generated by AI. They are resolved against semantically indexed, cryptographically signed components rather than executed directly.

2. WR Code, UI Mapping, and the Semantic Overlay Process

A central element of the architecture is the WR Code–enabled UI that assists publishers and users in mapping their websites to the internal model.

2.1 Semantic Parsing and Vector Storage

A structured parser analyzes the website and builds a rich representation including:

- full DOM hierarchy
- bounding boxes and geometric layout
- grouped regions (tables, lists, cards, sections)
- ARIA/role metadata and other semantic hints
- stable internal identifiers for all relevant elements
- embedding-based semantic descriptors stored in a vector database

This representation is the foundation for both human-authored and AI-generated augmentations.

2.2 WR Code Mapping UI

WR Code exposes a mapping interface that operates on top of this parsed representation. For each detected element, the system:

- generates a configuration form within the WR Code UI
- allows publishers and users to assign natural-language descriptions and synonyms
- defines overlay anchors, placement rules, and interaction constraints
- configures privacy and masking behavior
- stores all descriptors and mappings in a structured, semantically searchable form

The result is a tight coupling between the website's structure and its semantic description, which is crucial for high-precision overlay behavior.

2.3 AI-Generated Augmented Overlays in Real Time

The same parser and vector-encoded mapping also enable AI-generated Augmented Overlays in real time.

When a user or agent requests an augmentation (“highlight the most relevant risk indicators in this table”, “explain the fees in this section”), the orchestrator:

1. Interprets the plain-text request.
2. Resolves referenced elements and regions using the vector database and parser IDs.
3. Selects appropriate overlay components (e.g., badges, highlights, explanations).
4. Instantiates an Augmented Overlay template purely from these components.

No new code is generated; all behavior arises from mapping text instructions to pre-verified overlay components and previously mapped UI structures.

3. Mini Apps: Local Functional Units Without Data Sharing

Mini Apps represent structured, localized operations on device-resident or page-resident data. Templates specify:

- Permitted inputs: DOM extracts, metadata, local files (with explicit consent), device signals
- Operations: extraction, classification, comparison, validation, transformation
- Presentation: overlay widgets, summaries, contextual indicators

Mini Apps are assembled from verified functional components with strict input/output contracts.

A key benefit is that publishers can define their business logic once, and users can apply this logic to their own local data without any data leaving the device. Only the logic is distributed; the data remains private.

4. Workflows: Deterministic Multi-Step Logic

Workflows formalize multi-stage, deterministic processes including:

- fetch → validate → transform sequences
- normalization, aggregation, and scoring steps
- cryptographic verification (e.g., WRStamps, WR Code handshakes)
- conditional branches and gating logic

Workflow templates are realized via verified workflow units, rather than unrestricted scripting.

This allows publishers to distribute complex, business-specific workflows that:

- execute on the user's device
- respect privacy and consent boundaries
- remain cryptographically verifiable and auditable

Again, the data does not need to leave the user's environment; only the workflow logic does.

5. AI Agents: Constrained Behavioral Shells

AI Agents are specified as plain-text behavioral shells that define:

- Observation boundaries: which data or UI regions can be inspected
- Reasoning patterns: allowed analysis strategies, summarization modes, comparison methods
- Interaction policies: when to propose Mini Apps, Overlays, or Workflows; when to ask for confirmation
- Privacy constraints: which data must never be transmitted or exposed

Agents orchestrate existing components—UI, functional, workflow—with generating new code.

They can be refined through structured feedback loops to improve responsiveness, mapping accuracy, and alignment with user preferences over time.

6. Secure Component Libraries

The architecture relies on multiple cryptographically signed, semantically annotated libraries:

- **Code / Functional Library:** extractors, classifiers, scoring modules, validators, transformation operators
- **UI / Augmented Overlay Library:** visual primitives with constrained placement and behavior
- **Workflow Library:** deterministic multi-step routines, verification blocks, transformation pipelines
- **AI Agent Shell Library:** behavioral templates that define roles, reasoning modes, and autonomy boundaries

Each component is:

- described by precise semantic metadata
- indexed in vector space for retrieval
- bound by strict I/O and security contracts

Plain-text templates are resolved against these libraries, ensuring unambiguous, safe instantiation.

7. Auto-* Pillars for Real-Time User Support

On top of this foundation, the system supports adaptable, context-sensitive automation via four key pillars:

- **Auto-Mini-Apps**
Automatically suggested Mini Apps that adapt to the current page, data, and user intent, instantiated from templates in real time.
- **Auto-Augmented-Overlays**
Dynamically generated overlays that highlight, explain, or guide users through complex interfaces using the parser + WR Code mapping + vector semantics.
- **Auto-AI-Agents**
Behaviorally constrained agents that configure themselves based on templates, context, and user preferences, while remaining within verified boundaries.
- **Auto-Workflows**
Automatically assembled, context-aware workflows that chain verified units to support multi-step tasks without manual scripting.

These four pillars provide real-time, context-aware user support in workflows while preserving strict security and privacy constraints.

8. Adaptive Refinement and User Experience

Because all elements are:

- described by text,
- mapped through semantic embeddings,
- instantiated from verified components, and
- refined via feedback loops,

the system can mimic and refine real-time adaptation over time without ever executing dynamically generated code.

User corrections, publisher updates, and agent feedback improve:

- UI mappings,
- semantic descriptions,
- component selection, and
- behavior templates.

This leads to a user experience that approaches the fluidity of fully generative systems but remains deterministic, auditable, and privacy-preserving.

B. WR Code Registered Publishers

Publisher-provided mini apps must adhere to the same compositional model.

They are:

- **identity-bound to the publisher**
- **domain-verified**
- **publicly readable**
- **cryptographically tamper-proof**
- **pre-vetted before approval**

Because publishers operate within the same secure architecture, they can provide highly specialized mini apps while preserving full auditability.

3. Continuous Multi-Stream AI Analysis

Mini apps are not triggered by simple heuristics but by a continuous multimodal context feed.
The orchestrator provides the AI with a time-ordered stream containing:

- **DOM changes and structural updates**
- **user intent signals**
- **cursor patterns (hovers, selections, focus shifts)**
- **scroll and viewport dynamics**

- WR tag visibility and metadata
- explicit user commands
- optional OCR for non-DOM software

From the incoming intent stream, the AI constructs a *semantic blueprint* of the MiniApp that would best support the user's context.

This blueprint contains only **references to capabilities**—never executable code.

The orchestrator then resolves these references against the **verified, WR-stamped component library**.

If all required functional and UI blocks are present, the MiniApp is assembled deterministically by linking the pre-existing artifacts according to their declared interfaces.

No dynamic scripting or generated code is involved.

If a referenced capability does not yet exist in the library, the orchestrator records a **missing-block descriptor** (including semantic embedding and expected I/O) for later review and potential inclusion.

By combining semantic planning with a strictly verified component set, the system can provide precise, context-aware augmentation while ensuring that only trusted, pre-audited logic is ever executed.

4. Auto-Apps & Auto-Overlay: Local Real-Time Adaptation

With Auto-Apps or Auto-Overlay enabled, the orchestrator becomes highly reactive:

1. Analyze the multi-stream context continuously
2. Detect user goals or emerging opportunities
3. AI simulates a suitable mini app blueprint
4. Orchestrator assembles the app from verified components
5. Sandbox executes the composed mini app safely

All data processing remains fully local.

No personal data is transmitted externally.

All behavior remains deterministic, sandboxed, and inspectable.

The result looks and feels like “AI generating live code,” yet it respects all browser security constraints.

5. Direct User Commands for Fine-Grained Control

Users can refine or override the AI at any time.

For example:

- “Show the score bigger.”
- “Attach this analysis to WR Tag 32.”
- “Compare these two values.”

These commands translate into updated configurations, not new code.

The orchestrator simply regenerates the composed mini app using safe building blocks.

6. Augmented Overlay Follows the Same Secure Composition Model

The augmented overlay layer—highlighting, badges, embedded agents, dynamic panels—uses exactly the same mechanisms as mini apps:

- AI simulation →
- validated composition →
- cryptographic verification →
- sandboxed execution

This yields a coherent, trustworthy overlay environment with no risk of inline code, XSS vectors, or unverified logic.

Collaborative Feedback Loops & Swarm Intelligence: Human-in-the-Loop Problem Solving

The system supports more than individual real-time enhancements.

It enables a collaborative, domain-specialized feedback loop where groups of users and AI jointly explore complex topics—financial modeling, scientific reasoning, engineering analysis, or any structured decision problem.

The process is fully auditable and privacy-preserving.

1. Local AI + Domain Stream: Tailored Mini Apps for Each Participant

Each participant's orchestrator analyzes:

- research data
- financial tables
- scientific charts
- domain-specific models
- WR-tagged artifacts
- OCR-extracted content
- user goals and commands

The AI simulates a suitable mini app.

The orchestrator assembles it entirely locally using verified components.

No raw personal data, intermediate reasoning steps, or intent logs leave the device.

2. Collaborative Brainstorming Through Secure Output Sharing

Instead of sharing raw context or user data, each device sends only:

- structured outputs
- component-based explanations
- safe template configurations
- derived analytical results

These are streamed back into a WR Code-connected collaboration environment via DOM or OCR channels.

Proof of Execution Audit Layer

Every step is logged using a tamper-proof, cryptographically verifiable Proof of Execution chain, which records:

- which components were used
- how a mini app was composed
- which data bindings were applied
- which LLM evaluations contributed
- the exact processing sequence
- versioned provenance of all elements involved

This creates a reproducible, regulator-friendly chain of evidence without exposing private user data.

3. Multi-Model Evaluation: Ranking Analytical Contributions

The collaboration platform evaluates submissions using:

- multiple LLMs
- independent scoring heuristics
- domain-specific knowledge checks
- cross-model consistency verification

Because all outputs originate from signed components and audited composition steps, each analytical contribution is:

- **reproducible**
- **accountable**
- **cryptographically verifiable**
- **inspection-ready for regulated domains**

4. Swarm-Style Intelligence Feed

Highly rated results flow back into a shared feed:

- **optimized analytical methods**
- **improved mini-app configurations**
- **alternative solution paths**
- **multi-step reasoning patterns**
- **domain-specific heuristics**

Users can immediately trigger:

- **Iterate**
- **Apply to my data**
- **Refine**
- **Generate alternative composition**

- Compare
- Extend

All results remain fully governed by Proof of Execution, forming a continuous, tamper-proof learning system.

5. Human-in-the-Loop Machine Learning Cycle

The loop functions as follows:

1. Users explore data or research content
2. Local AI proposes mini apps (simulation → composition)
3. Users refine or steer the analysis
4. Findings are shared back via WR Code
5. Multiple LLMs evaluate and rank results
6. High-quality methods flow back to the shared feed
7. Users iterate

Proof of Execution ensures that the entire cycle is:

- traceable
- auditable
- reproducible
- regulator-compliant

No private data needs to be shared, and all reasoning steps remain inspectable without exposing user context.

TimesDesk as the Initial Implementation by Optimando AI

TimesDesk, a project of Optimando AI, will be the first operational platform to deploy this architecture.

It applies the model to financial analytics, comparative modeling, and asset exploration—demonstrating how:

- **secure composition**
- **multi-stream analysis**
- **collaborative feedback loops**
- **swarm-style evaluation**
- **tamper-proof orchestration**
- **and LLM-guided real-time adaptation**

...can operate within a unified, privacy-preserving system.

This deployment does not claim global novelty; it simply represents the initial implementation within the Optimando AI ecosystem.

After validation in TimesDesk, the framework can be extended to:

- **scientific research environments**

- engineering and simulation workflows
- academic and policy evaluation
- legal analysis and reasoning
- complex multi-factor planning
- any domain requiring structured human-AI cooperation

The architecture is domain-agnostic and designed for secure scaling.

Dynamic Template Composition for Multi-Agent Workflows and Mini-Apps

Both mini-apps and multi-agent workflows follow a unified, template-based execution model. Each component—whether it is a UI-driven mini-app, an analytical module, or a multi-agent sequence—is defined through a declarative template describing roles, rules, allowed tools, and security parameters.

To make these templates flexible yet controlled, the system uses a three-tier composition model:

1. Publisher Templates (verified standard definitions)

These are officially provided, versioned, and cryptographically secured templates.

They define validated multi-agent workflows and mini-app logic and serve as a stable, reproducible foundation for widely used operations.

2. User Templates (local, personalized extensions)
Users may extend Publisher Templates or provide their own.
All modifications remain strictly validated and are limited to a parameter space that reflects security, compliance, and execution constraints.
3. Live-Generated Plans by AI (context-dependent runtime composition build from a library)
For novel or complex tasks, the system can generate additional template elements at runtime—either for multi-agent logic or for mini-app behaviors.
These live fragments are fully schema-validated and follow the same safety and policy restrictions as static templates. Users may optionally save these generated plans as their own templates.

A central Composer layer merges these three sources into a single deterministic execution plan. Conflicts are resolved along a fixed priority order (System Policies → Publisher → User → Runtime Additions). This allows both mini-apps and multi-agent workflows to adapt dynamically to user intent and context while remaining transparent, auditable, and predictable.

Extension to Augmented Reality and Physical Interfaces

The principles of the Augmented Overlay extend beyond digital screens into **augmented reality (AR)** environments, enabling seamless interaction with physical devices, control panels, and printed materials. Through the **WR Code Protocol**, each physical element—whether a machine, keyboard, tool, or instrument—can be *registered once* and linked to a precise, verifiable digital overlay. This

allows users to receive interactive, context-aware guidance and execute backend automations directly through natural gestures or voice commands.

Each registered object includes a structured metadata profile defining its geometry, functions, and semantic relationships (e.g., “power switch,” “safety valve,” “Enter key”). Using **open source AR technologies** such as **AprilTags**, **OpenCV**, or **ARToolKit**, the overlay can accurately recognize and anchor virtual elements in 2D or 3D space, ensuring consistent alignment between digital instructions and real-world components. Hybrid tracking—combining fiducial markers for reliability with feature-based detection for flexibility—provides robust recognition in industrial, office, or educational environments.

Once recognized, the overlay transforms physical interaction into an intelligent, conversational process. The **pointer or cursor**—whether projected from a headset, tracked by camera, or controlled via a physical device—becomes the universal interface. Users can simply point to an element and ask, “*Do I need to click here now?*” or “*What happens if I turn this knob?*”

The system interprets gestures and questions in context: it retrieves current procedure steps, historical memory, and relevant metadata to respond precisely—without requiring the user to specify which element they meant or what step preceded it. This context-anchored reasoning makes interaction natural, transparent, and error-resistant.

Such pointer-based interactions can also **trigger backend automation workflows**. For example, pointing at a machine’s control knob and saying “*Log maintenance done*” could automatically register completion in a connected n8n or MCP workflow, update maintenance records in

PostgreSQL, or notify a supervisor. These automations are defined in the WRCode metadata and executed locally or via secure cloud connectors—ensuring traceability without exposing private data.

Augmentation itself is **multimodal** by design. Depending on the context, the overlay can combine **visual, auditory, and haptic feedback** to guide the user effectively. A shared **library of open visualization elements**—including arrows, glow effects, outlines, animated trajectories, or spatial indicators—ensures consistent, comprehensible guidance across all environments.

These elements are standardized, reusable, and openly available, allowing communities to contribute new visualization types. For example, a glowing arrow can highlight the next button to press, a pulsing outline can indicate system readiness, and an animated path can show the correct motion for aligning or tightening a part. When combined with subtle audio and haptic cues, the overlay adapts dynamically to user behavior and environmental conditions.

The system supports a wide range of **input modalities**. In addition to traditional camera-based tracking, it can integrate with **EMG-based wristbands** such as Meta’s neural wristband, **finger-tracking sensors, finger-mouse controllers, and hand-gesture interfaces**. These devices detect muscle impulses or micro-movements, allowing users to interact with the overlay using subtle gestures—pointing, pinching, or tapping in mid-air. Such technologies complement the pointer concept, offering hands-free, low-friction control well suited for AR glasses, industrial safety environments, or field operations.

Manufacturers do not need proprietary software. They simply register their devices once under the WR Code Protocol, providing metadata and anchor definitions. The open, interoperable overlay engine interprets this data uniformly across smartphones, AR glasses, or desktop environments.

Users and technicians can **extend the knowledge base collaboratively**, adding annotations, images, or voice notes directly to overlay elements. These contributions are semantically indexed so future users can query, “*show previous fixes for this control*” or “*explain the calibration history of this device.*”

The same framework also enhances **printed media**—books, manuals, or magazines—using image-based recognition to attach contextual overlays. Readers can point to a diagram and ask, “*Can you show me how this works?*”, prompting visual animations, video clips, or voice-guided explanations rendered directly on the page.

All processing follows a **local-first, privacy-preserving architecture**: recognition, reasoning, and visualization run on-device for speed and confidentiality, while cloud augmentation remains optional for collaboration or compute-intensive analysis.

By combining **AprilTag precision, OpenCV flexibility, multimodal visualization, and next-generation gesture interfaces**, the Augmented Overlay unifies digital and physical interaction. It turns machines, keyboards, and documents into transparent, explainable, and adaptive interfaces—bridging human intuition with real-time automation in a truly open, extensible ecosystem.

Extension to Robotics and Autonomous Agents

The integration of robotics into the **Augmented Overlay** ecosystem builds upon the same foundational principle: a shared, standardized understanding of context through the **WR Code** Protocol. In this vision, robots become not just executors of pre-programmed commands but active participants within the same augmented environment that humans perceive.

To operate within this ecosystem, manufacturers would adapt their systems to the **WR Code** Protocol. This requires embedding a **WR Code** scanner and a context interpreter capable of reading the augmented overlay and understanding the associated metadata. Once implemented, any compliant robot can recognize the augmented elements, interpret their semantic meaning, and interact with them intelligently — without requiring proprietary APIs or specialized software integrations.

When a robot scans a registered **WR Code**, it retrieves a structured metadata profile describing the physical device or environment: geometry, control elements, and contextual logic. The Augmented Overlay provides additional real-time cues — such as position anchors, arrows, or procedural hints — that can be interpreted both visually and semantically. Through this unified framework, the robot understands not only *what* elements exist but also *how* they relate to operational tasks and safety conditions.

For example, a service robot scanning the **WR Code** of a dishwasher could instantly identify control buttons, read the overlay's procedural context, and perform the correct steps to start or

stop a cycle. All necessary information is provided through the standardized **WR Code** and overlay data — no manufacturer-specific software or cloud service is required.

A key benefit of this approach is demonstration-based learning. Users can *teach* robots directly by showing how tasks are done. The orchestrator observes and records these demonstrations, capturing positions, gestures, and contextual annotations from the overlay. AprilTags, OpenCV tracking, or similar fiducial systems ensure accurate spatial alignment, enabling the robot to later reproduce the same actions with precision.

The same **WR CODE** Orchestrator infrastructure — built on PostgreSQL and LangGraph — can record and document each interaction step as structured data. This enables iterative improvement: users can review past sessions, adjust contextual definitions, or refine the metadata of **WR Code**—registered devices to make the workflow more efficient and adaptable to specific environments.

In larger machine parks or industrial environments, the orchestrator can manage multiple robots operating under this protocol. Robots can alternate between a replica (simulated) environment and the real workspace, both of which use the same **WR Code** anchors and overlay definitions. This allows testing, training, and verification in a controlled environment before execution on the factory floor.

An optional conceptual extension involves integrating transparent reasoning and camera feed visualization through the orchestrator. This manufacturer-dependent enhancement would allow operators to monitor what the robot “sees,” review its reasoning process, and adjust the overlay or context definitions in real time. Such visualization layers are not required for the **WR Code** Protocol itself but represent a future path toward fully explainable robotics and seamless human–machine co-learning.

In essence, the **WR Code**–enabled **Augmented Overlay** creates a universal interface language between humans, machines, and robots. Manufacturers who adopt the protocol can ensure that their systems understand standardized contextual metadata, anchor definitions, and procedural logic — enabling interoperability across brands, environments, and applications.

Through this approach, robots no longer require isolated, closed control systems. Instead, they become contextual participants in an open, explainable, and continuously improving automation ecosystem — capable of understanding, reasoning about, and adapting to the same augmented world that humans experience.

Spatial Grouping and Hierarchical Context Models

The WR Code Protocol can be extended beyond single-object registration to represent entire physical environments as hierarchical, context-aware structures. In this model, each space—such

as a kitchen, laboratory, or factory hall—is represented by a **root or “roof” WR Code**, which serves as the anchor for all subordinate (“leaf”) WR Codes within that environment.

Roof WR Codes as Spatial Anchors

A roof WR Code defines the boundaries, geometry, and semantic map of an environment. It encapsulates a **floor plan and 3D spatial model** that can be generated directly by end-users through simple capture and annotation workflows. Using a smartphone or AR headset, users can photograph the environment and tag objects with natural-language descriptors such as *“This is the fridge,”* *“Coffee machine expected here,”* or *“Sink located in this zone.”*

In many cases, devices such as coffee machines or smaller appliances are not fixed but **placed within an expected zone rather than a fixed coordinate**. The annotation therefore defines an **anticipated spatial area**—a bounding region that indicates where such an object is typically located. During operation, the robot or AR system uses its **onboard sensors and cameras** to refine the precise position within that area, aligning the detected object with the annotated expectation. This hybrid approach combines human-level spatial labeling with machine-level perception, achieving high localization accuracy without requiring full manual mapping.

These annotated captures are processed by a **blender-based converter pipeline**, which reconstructs the 3D geometry and exports it into open simulation formats—such as **USD**

(Universal Scene Description), glTF, or other widely accepted standards compatible with NVIDIA Omniverse, Unity, and Unreal Engine. The result is a verifiable, semantically enriched digital twin anchored to a single WR Code.

Each roof WR Code may also include a **fiducial marker** (e.g., AprilTag) or **spatial hash reference** to ensure accurate alignment between physical and digital coordinates. This allows AR devices, cameras, and even autonomous robots to instantly localize themselves within the known environment using a consistent, open standard—without relying on proprietary SLAM datasets or cloud-locked mapping APIs.

Crucially, **scanning the roof WR Code automatically loads all subordinate WR Codes within that spatial group.**

Whether performed by a humanoid robot, an AR headset, or a mobile device, this single scan retrieves the full environment structure—its 3D layout, all associated device overlays, and contextual metadata. The user or robot gains an immediate, coherent understanding of the environment: the spatial positions of objects, their functions, and their operational context. This drastically reduces setup complexity and ensures interoperability across heterogeneous systems.

Leaf WR Codes and Contextual Inheritance

Beneath the roof layer, **leaf WR Codes** represent individual machines, tools, or control elements within the same environment. Each leaf WR Code carries its own **Augmented Overlay dataset**, including:

- Contextual metadata (functions, safety constraints, operational logic)
- Visual overlays and interaction affordances (buttons, arrows, trajectories)
- User manuals, maintenance logs, or manufacturer-provided guidance

Leaf WR Codes **inherit** the spatial and semantic context of their parent roof WR Code, ensuring that all devices share a common coordinate system and procedural graph. This hierarchical grouping enables both humans and robots to reason about complex environments as structured, self-describing datasets—similar to a file system for spatial intelligence.

Collaborative Environment Mapping

Both manufacturers and end-users can contribute contextual data to any WR Code layer. Manufacturers may publish verified overlay templates, calibration data, and operating procedures, while users can attach localized annotations, experience reports, or corrective notes. Each contribution is cryptographically signed and anchored (e.g., via IOTA or Solana), ensuring **traceability, authenticity, and incremental versioning**.

In multi-user scenarios, these spatial hierarchies become **collaborative knowledge spaces**: multiple technicians can work within the same WR Code environment, augmenting it with new device registrations or updated 3D models. The orchestrator synchronizes these changes locally, allowing offline or private deployments while maintaining global verifiability.

Simulation and Robotic Training Integration

Because roof WR Codes store geometry and contextual semantics in standard 3D formats, entire environments can be imported directly into simulation engines.

Humanoid or service robots can then **train inside these digital twins**, using the exact layout, object dimensions, and control logic of the corresponding real space. The same WR Code anchors that guide human users in AR also serve as **training references for robotic vision and manipulation models**.

Through this unified approach, a robot trained in a simulated kitchen can later operate in the physical counterpart with **minimal recalibration**, as all spatial anchors and metadata remain consistent between the digital and physical representations.

Dynamic Expansion and Interoperability

New devices can be added at any time by registering additional leaf WR Codes under the existing roof WR Code. The orchestrator automatically integrates their overlays, control definitions, and metadata into the shared environment graph, updating both the 3D digital twin and the procedural automation layers.

This **modular spatial taxonomy** allows environments to evolve organically—supporting homes, factories, or laboratories that continuously expand and adapt. Every object, from a coffee machine to an industrial lathe, becomes part of a **linked, verifiable knowledge network** that bridges augmented reality, automation, and simulation.

In summary, the hierarchical WR Code model transforms environments into living, self-documenting ecosystems. By combining user-generated spatial capture, open 3D standards, and verifiable metadata inheritance, it creates a scalable framework where **humans, robots, and AI systems share the same contextual understanding of space**—locally, privately, and interoperably across both virtual and real-world domains.

Roof Scan Discovery and Hierarchical Loading Process

When a robot or AR device performs a **roof scan**, it initiates a hierarchical discovery process that reconstructs the complete spatial and semantic context of an environment from a single point of reference.

The scan begins with optical recognition of the **roof WR Code**, which encodes a unique environment identifier (WRID) and a **verification hash** linking to its registry record.

This record can be resolved either **locally**—from data embedded in the user’s own system—or **remotely** via the **distributed registry and optional public vector infrastructure at wrcode.org.**

Once verified, the orchestrator loads the **environment manifest**—a structured dataset (JSON / YAML) that defines

- the **hierarchical topology** (roof → zones → leaf WR Codes),
- the **geometric model** (floor plan, coordinates, 3D assets in USD or glTF),
- the **semantic context** (functions, dependencies, workflows, safety logic), and
- the **integrity proofs** (WRStamps, version lineage, authorship metadata).

The WR Code ecosystem uses a **hybrid storage model**:

not all data are embedded into vector databases.

Instead, each data type is stored in the most appropriate technical format:

- **Geometric and structural data** (3D models, CAD blueprints, spatial coordinates) are stored as standard files or in spatial databases optimized for precision and rendering performance.
- **Procedural logic and machine states** may reside in relational or graph databases to support deterministic execution and version control.
- **Contextual and unstructured data**—such as user manuals, maintenance notes, descriptions, safety guidelines, or operational narratives—are **embedded as vectors** within a local or distributed **vector database** (e.g., pgvector, Qdrant, or FAISS).

These vectorized entries form the **semantic layer** of the environment, enabling **contextual reasoning** for both humans and autonomous agents.

Each WR Code contributes its own embedding set, representing its descriptive content, associated procedures, and historical annotations.

Together, these embeddings create a **semantic knowledge space** that can be queried through similarity search and retrieved by local or connected large language models (LLMs).

When a user or robot asks, “*How do I calibrate this valve?*” or “*What does this indicator mean?*”, the orchestrator performs a **semantic retrieval** over the relevant vector database—either the **locally embedded dataset** or the **verified public dataset hosted at wricode.org**. This allows the LLM or reasoning agent to respond precisely, grounded in the contextual meaning of the environment, rather than relying on static instructions or fixed templates.

For **offline or private installations**, all reasoning and retrieval take place locally, ensuring data sovereignty and confidentiality.

When connectivity is available, the orchestrator can optionally query **public WR Code vector databases** on wricode.org, which contain **verified, openly shared embeddings**—for instance, official manufacturer manuals, safety guidelines, or community-contributed annotations.

Each remote vector entry is cryptographically anchored, ensuring provenance and tamper-proof synchronization across installations.

During initialization, the orchestrator constructs a **spatial-knowledge graph** that fuses traditional geometric data with semantic embeddings.

Zone-based objects—such as a coffee machine typically placed “within this area”—are represented as **probabilistic nodes**, where the robot uses onboard sensors (camera, LiDAR, or depth systems) to identify and align real-world objects with their corresponding metadata and context embeddings.

This hybrid reasoning model ensures both **geometric precision** and **semantic understanding**.

Once validated, the orchestrator activates the appropriate **Augmented Overlay layers**, enabling interactive guidance, AR visualization, and automation triggers (e.g., n8n or MCP workflows). All reasoning steps, actions, and updates are **WRStamped**, preserving full traceability and integrity.

In this architecture, the **vector database—local or hosted at wrcode.org**—serves as the **semantic substrate** for contextual understanding, while geometric, procedural, and operational data remain stored in their optimal native formats.

The WR Code registry itself acts as the **integrity and linkage layer**, unifying these heterogeneous data types under one verifiable and interoperable protocol.

By scanning a single **roof WR Code**, any compliant robot, AR headset, or orchestrator instance gains immediate access to a **multi-modal world model**—combining geometric precision, procedural logic, and semantically retrievable knowledge.

This architecture ensures that both humans and machines can share, reason about, and act within the same contextual framework—securely, verifiably, and efficiently across both local and distributed environments.

Privacy and Security Implications

Importantly, all logic and transformations would occur exclusively on the user's machine. The feature is meant to aid local inspection and debugging. As such, it aligns with strict privacy requirements (e.g., GDPR, HIPAA, and industry-specific compliance standards).

Development Status

This feature is currently a conceptual proposal and will not be part of the initial release. It represents an advanced step toward interactive AI debugging, and its feasibility, usability, and performance implications must be carefully evaluated before full implementation.

Nonetheless, the Augmentation Overlay aligns with the broader vision of giving end users greater control, visibility, and confidence when interacting with masked data and semi-autonomous agent systems.

Use a Dedicated Virtual Machine (VM) or Isolated Workspace

It is recommended to run the orchestration environment (including Master Tab, Coordinator, and agents) inside a dedicated virtual machine or isolated operating environment.

This provides two core benefits:

1. **Technical Isolation** – Prevents cross-contamination of session data, cookies, or clipboard content between personal activity and AI workflows.

2. **Contextual Awareness** – Using a VM helps users mentally distinguish between “AI mode” and regular activity, reducing the risk of unintentionally exposing sensitive data. The separation acts as a continuous reminder that interactions may be processed, embedded, or analyzed — and should be treated accordingly.

Use a Separate Browser Profile or Session

For users not using a full VM, it is advised to run *optimando.ai* in a dedicated browser profile or container session. This limits access to personal cookies, autofill data, and login states that may otherwise be unintentionally exposed to AI agents or helper tabs.

Masking Defaults and Manual Overrides

Users should:

- Review and configure default masking rules (names, addresses, numbers, etc.) before running automation tasks.
- Use manual overrides only when necessary — e.g., for document reasoning tasks that require semantic fidelity.

- Understand that masking only applies within agent-controlled flows and does not affect direct input into third-party AI systems (e.g., public chatbots).

Avoid Manual PII Entry in Master Tabs Connected to AI-Services

The Master Tab is the entry point to many workflows. While it is architecturally isolated from helper agents, it may still be connected to external systems (e.g., websites, LLM chat interfaces).

Users are responsible for avoiding direct entry of personal, financial, or sensitive data into services where masking cannot apply.

Prefer Local or Air-Gapped Modes for High-Sensitivity Tasks

When working with proprietary, legal, or highly sensitive data, users should:

- Prefer offline LLMs or local inference engines
- Disable or restrict outbound AI service calls entirely
- Inspect agent logic manually before execution

🔧 The Future of Personal Infrastructure — Intelligent. Private. Local.

Our browser-based orchestration system unifies **digital workflows, smart environments, and AI-driven automation** — all locally hosted, privacy-respecting, and fully under the user's control. No cloud dependency. No vendor lock-in.

While traditional automation relies on fixed routines — like "*doorbell rings → show camera feed*" — the real breakthrough lies in the system's **built-in optimization layer**. It goes beyond basic cause-and-effect logic, enabling context-aware decisions that dynamically adjust both digital workspaces and connected smart home devices in real time.

To enable these physical interactions — such as adjusting lighting, blinds, or accessing LAN-only camera streams — users can integrate any browser-accessible **smart home platform**. A popular and well-suited option is **Home Assistant**, an **open source** system that runs locally, offers a powerful browser interface, and communicates securely via local APIs. It fits seamlessly into the orchestration layer's architecture, while maintaining full flexibility and user control.

⌚ Real Optimization in Action — With Smart Home Integration

☒ Competing Priorities, Managed Intelligently

You're in a deep work session. The doorbell rings, your child makes noise, and an AI-generated report begins processing.

→ The system allocates screen space to the camera feed, defers the baby monitor unless noise continues, and queues the report for silent review — without disrupting your flow.

Environmental Context Shaping

As you begin focused writing at dusk, the system dims lights, closes blinds via Home Assistant, and suppresses non-essential automations — keeping the mental environment aligned with the task.

Scenes That Evolve With You

Start “Relax Mode”: lights soften, blinds close, music plays. Leave the room, and the system pauses playback, powers down smart lights, and waits until you return.

→ Scenes respond to presence — not just preprogrammed steps.

Smart Family Awareness

A child stirs late at night.

→ If you’re watching a movie, the LAN-only baby cam overlays discreetly. If you’re on a call, it appears silently on a secondary display.

→ One event, context-aware outcomes based on your current activity.

A Flexible, Controllable Optimization Layer

The optimization layer is not rigid or opaque. Users can:

- Toggle it off at any time

- Override it manually via voice or keyboard commands
- Replace it for specific scenarios where deterministic behavior is preferred

It is also intended to learn over time, storing preferences and command patterns locally — adapting its logic based on user intent and feedback, while keeping all data fully private.

For professionals managing business, home, and privacy-critical environments, this offers something truly new: A control layer that adapts like a real assistant — across screens, smart homes, and time.

The software components of the orchestration system—including the OpenGiraffe orchestrator, the Augmented Overlay engine, WR Code integration, WRVault, and WRGuard—are published under the **OpenGiraffe Protected License (OGPL-1.0)**. This is a **source-available** license that allows inspection and Template development while prohibiting redistribution, SaaS hosting, code forking, or removal of embedded system branding.

The **conceptual framework**, including the orchestration model, architectural diagrams, workflow logic descriptions, and all explanatory documentation presented in this paper, is licensed under the **Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0)**. This ensures that the conceptual layer remains open, traceable, and preserved as publicly verifiable prior art.

This dual-license structure—**OGPL-1.0 for the software and CC BY-SA 4.0 for documentation**—provides strong protection for runtime components while enabling open, collaborative use of the underlying ideas and published research.

For commercial, closed-source, white-label, or enterprise deployments requiring alternative terms, including custom attribution or branding agreements, a separate **commercial license** is available.

WR CODE: A Comprehensive, Future-Proof Security Framework for Local AI Orchestration in High-Risk Environments

Abstract: **WR CODE** is a source-available, modular security framework designed to enable fully local execution of AI-powered workflows while ensuring tamper-proof access control, immutable logging, and strict separation of data, roles, and configurations. The system is built specifically for high-risk environments where data integrity, long-term security, and operational resilience are paramount. The architecture is designed to prevent not only current cyber threats but also future risks posed by emerging technologies such as quantum computing.

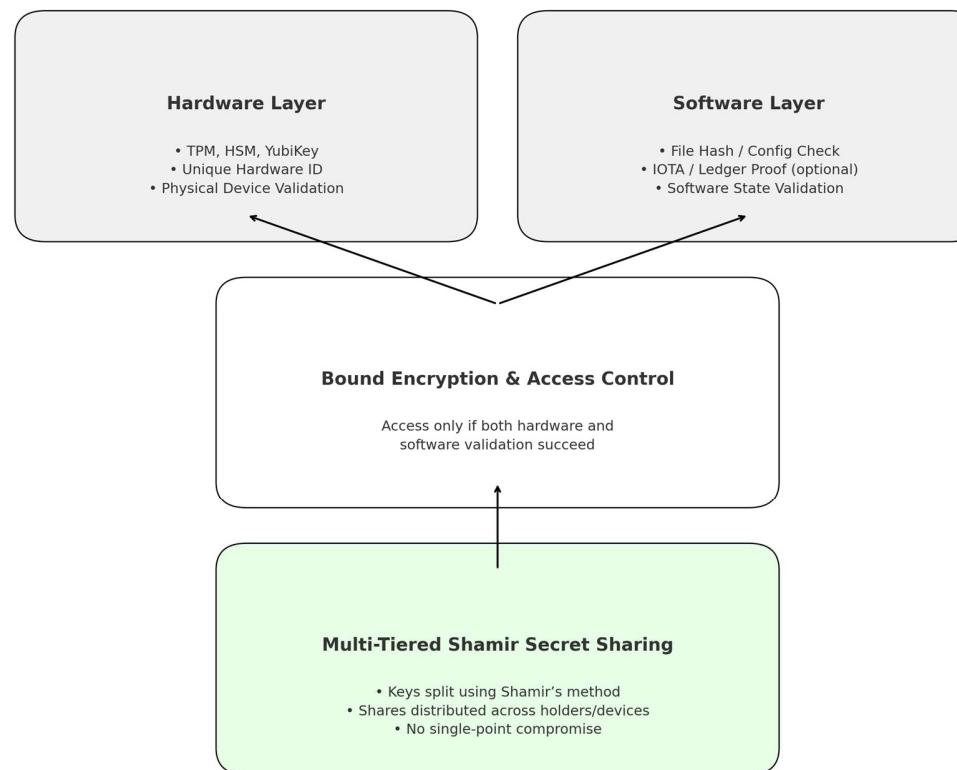
1. Purpose and Design Philosophy:

The rise of AI and Large Language Models (LLMs) in sensitive business processes introduces new vulnerabilities. **WR CODE** exists to close these gaps by offering a framework where no unauthorized user can copy, decrypt, or misuse AI data—now or in the future. Even if encrypted containers are exfiltrated, their contents remain unusable due to:

- By binding encryption to both hardware and software conditions—such as cryptographic file hashes or Solana/IOTA-based proofs—tamper resistance is enforced across physical and digital layers.
- Multi-factor authentication

- Ledger-based methods (blockchains, DLTs) provide integrity in connected systems, while air-gapped environments achieve comparable integrity properties through local cryptography and secure hardware (e.g., HSMs, TPMs).
- Cryptographic configuration freezing
- Revocable key material and user access

Hardware- and Software-Bound Encryption with Shamir Secret Sharing



2. Role-Based Data and AI Separation:

WR CODE allows every employee to access the same locally hosted AI engine while strictly controlling:

- **The data each role can access (through separate vector databases in encrypted containers)**
- **The AI templates, prompts, workflows, and configurations presented to them**

This ensures that a researcher, HR staff, and finance controller each interact with the AI differently—both in what they can do and what information they can access. Department-specific containers keep sensitive information compartmentalized. The Locker App authenticates users, securely transfers role identifiers, and ensures only authorized configurations are loaded. User permissions can be revoked instantly, disabling all access in real time.

3. Advanced Security Layers (Optional):

WR CODE supports multiple layers of defense, which can be combined to match an organization's risk profile:

- Device Fingerprint Binding: Ensures execution only on registered hardware.
- Multi-Factor Authentication (MFA): Adds OTP-based secondary authentication.
- Hidden and Departmental Containers: Strict separation of data, templates, and workflows for each department.
- Role-Based AI Customization: Ensures prompts, outputs, and automation are aligned to user roles.

- Tamper-Proof Configuration Freezing: Cryptographic hashes ensure that once a security configuration is deployed, it cannot be altered without detection. Any mismatch triggers immediate protective measures.
- Mandatory Logging with Auto Shutdown: In high-risk settings, every action must be logged—locally, to a management PC, or immutably to a ledger. If logging is not possible, **WR CODE** automatically locks down or shuts down to prevent any untraceable execution.
- Immutable Attempt Logging and Revocation: All events, including failed login attempts, role escalations, and configuration changes, are logged immutably. User access can be revoked instantly through ledger or network-triggered commands.
- Tiered Shamir Secret Sharing (SSS): For critical operations or emergency overrides, Shamir Secret Sharing ensures no single user can act alone. Multi-party authorization is enforced for accessing top-level containers, altering system settings, or releasing sensitive logs.

4. Hosting and Deployment Flexibility:

WR CODE is hardware-agnostic and can be deployed:

- On local devices
- In on-premises secure server environments
- In private enterprise clouds
- In fully air-gapped deployments or in mixed environments with immutable rule sets

It is scalable from small teams to large enterprise environments with multiple departments.

5. Logging, Auditability, and Compliance:

The system enforces:

- Full traceability of every access, action, and change
- To ensure tamper-proof logging, secure storage can be implemented locally, across networks, or via blockchain/ledger solutions. On management devices, secure elements such as HSMs, TPMs or embedded chips can further protect logs from manipulation, even in isolated environments.
- Automated detection and enforcement of system lockdown if logging fails or is tampered with

All activities are cryptographically verifiable, providing strong compliance support for industries under strict regulation.

6. Ledger Anchoring, Template Authenticity & Zero-Knowledge Proofs via Solana/IOTA

WR CODE is not just built for today's security challenges. Its design anticipates future threats, including quantum decryption attacks. Even if encrypted containers are stolen, they cannot be decrypted without the original hardware, valid key material, and verified configuration. The WR CODE project is committed to ongoing research into quantum-proof cryptography, Zero-Knowledge Proofs, and AI governance to ensure long-term security and innovation leadership.

WR CODE optionally integrates Solana/**IOTA**—a scalable, feeless distributed ledger designed for verifiable, tamper-proof anchoring of events, code artifacts, and execution traces without exposing sensitive data.

By selectively anchoring cryptographic hashes of:

- WR Code payloads
- Geofence presence proofs
- AI agent decisions
- Orchestration templates and automation scripts
- Workflow execution snapshots
- User activity events (where compliance requires)

WR CODE enables: Verifiable claims without centralized trust

- Tamper-proof logging of AI-driven actions and template integrity
- Immutable timestamping of sensitive events
- Optional privacy-preserving user activity audits

Why IOTA?

- **Feeless Transactions:** Scalable even for high-frequency micro-events.
- **Tangle Architecture:** Energy-efficient, lightweight, and parallelizable.
- **Rich Metadata Support:** Ideal for anchoring multi-agent decision chains and human-in-the-loop events.

- **Future-Ready for Post-Quantum Security:** Supports hash-based quantum-resistant signatures without protocol disruptions.

This broader use of Solana/IOTA within **WR CODE** extends beyond just **WR Code** authenticity. It provides the foundation for:

- Verifiable **template authenticity:** Ensuring that orchestrated actions stem from trusted, unchanged templates.
- Confirmable **code originality:** Proving that automation logic has not been altered or manipulated.
- Transparent **logging of AI decisions and outputs:** Allowing both machine and human participants to audit past actions.
- Optional **user activity logging:** With strict privacy controls, providing tamper-proof documentation of who initiated which actions in sensitive environments.

To encourage adoption, **WR CODE** promotes the idea that **WR Code** providers and enterprises anchor their hashes on Solana/IOTA, giving end-users and regulators the ability to independently verify authenticity, originality, and integrity. This could help create an ecosystem of higher trust without relying on centralized control structures.

However, while a cryptographic hash secures the integrity of any payload, true identity verification of the publisher still requires anchoring via identifiable wallets or decentralized identities (DIDs). This dual approach—content integrity plus source authenticity—allows for full trust without sacrificing privacy.

Unlike traditional blockchains such as **Bitcoin, Ethereum, or Solana**, IOTA's architecture is inherently more adaptable for post-quantum upgrades. Its feeless nature also makes it practical for the high volume of micro-interactions typical in AI orchestration.

Integration Scenarios:

1. **QRGiraffe:** Secure WR Code processing with optional IOTA logging for auditability.
2. **Geofence Proofs:** Anonymous presence proofs with optional ledger anchoring.
3. **AI Decision Trails:** Verifiable record of orchestrated AI-driven actions, including template verification.
4. **Optional Future Incentives:** Crypto rewards, NFTs, or smart contract automation where applicable.
5. **User Consent & Activity Logging:** Tamper-proof logs of user-triggered actions for compliance-heavy sectors.

Sensitive data is never stored—only cryptographic proofs are shared for independent verification.

7. Optimando Service Offering:

While WR CODE is freely deployable as Source-available, Optimando provides:

- **Expert consulting on AI orchestration security**
- **Customized role and container design**
- **Deployment of secure, tamper-proof environments**

- Hands-on staff training and compliance support

8. Conclusion:

WR CODE provides a holistic, future-proof solution for secure AI deployment. With role-based separation, immutable configuration, hardware-tied encryption, mandatory logging, and system lockdowns, it ensures that AI can be safely used in the most sensitive environments. Through optional advanced security features and **Optimando's** consulting support, organizations can adopt AI with confidence—without compromising on privacy, operational integrity, or long-term data protection.

WR CODE and its tools are source-available and trustless—enabling everything from easy setups to tamper-proof enterprise solutions. Future-ready LLMs without blind trust.



Verifiable Real-Time Automation with WRCode & IOTA For IOT-Devices

WRCode is a modular, source-available trigger and verification framework designed to anchor real-world events and automate decision-making with human-in-the-loop support. The system enables individuals and organizations to build highly adaptable automation workflows using locally run AI agents, verifiable templates, and privacy-preserving orchestration.

In fragmented environments—ranging from smart factories to legacy systems—there is often no standard infrastructure for coordinating, verifying, or recording complex technical workflows. WRCode addresses this by offering a **machine-readable orchestration trigger** that links a scanned WRCode to signed, auditable templates and context-specific logic.

Upon scanning, the WRCode system references a **WRStamped library**—a curated, tamper-proof collection of AI templates and agent instructions. These templates can be issued by manufacturers or overwritten by users through simple natural language commands. **Overrides are stored locally and take precedence** when available. If a manufacturer does not provide any WRCode for a specific machine, a WRCode developer can create a custom WRCode instance to integrate that legacy device into the system. This allows field technicians and domain experts to adjust automation behavior to their needs—even for unsupported equipment—without writing code.

The orchestrator dynamically loads and runs the required logic, which may include downloading tamper-proof, auditable code libraries and tools that are cryptographically anchored and verified. These libraries extend the orchestrator's capabilities to handle a wide variety of scenarios with minimal setup, enabling dynamic adaptation to the context of each WRCode scan. Companies providing domain-specific automation logic must adhere to the same verifiable source-available anchoring rules, ensuring all contributed code is fully auditable and tamper-proof. This is a core principle of the system's trustless security model, which by design does not permit hidden or unverifiable execution paths. This modular approach allows seamless integration of custom AI-driven workflows tailored to specific equipment, environments, or support needs. Examples include:

- Capturing photos, videos, or audio logs
- Submitting speech or text-based reports
- Executing device- or error-specific diagnostics

- Interfacing with live APIs exposed by compatible machines

Each step is documented in full. This includes:

- WRCode ID, Project ID, TemplateMap ID, Technician ID, Group ID, Timestamp
- Captured sensor data and human input
- All decisions made by the AI agents

This comprehensive interaction record is defined as a **Proof of Execution (PoE)**. It is structured using Merkle tree logic and **anchored on IOTA**, providing a scalable, immutable, and audit-friendly validation layer.

Unlike traditional blockchain systems, IOTA provides:

- Zero-cost, high-frequency anchoring
- Offline operation via edge-device queuing
- Energy efficiency suitable for embedded hardware
- Long-term immutability without exposing sensitive data

Solana complements the system by handling real-time access control. Group roles, permissions, and scan validation operate via signed Solana-based credentials, enabling fast response times, while IOTA serves as a long-term verifiability anchor.

The WRCode.org registry maintains templates, signatures, and developer roles. Codes issued by developer accounts are color-coded red and marked as internal-use-only. While most workflows are adaptable and overridable, **code governing peer-to-peer exchange and vault logic remains fixed** to maintain trustless integrity.

To enable deep contextual support, users can embed documents like user manuals or field instructions directly into the orchestrator's **local LLM layer**—allowing real-time assistance without cloud dependencies.

This approach is not limited to IoT. WRCode + WR CODE is a **general-purpose orchestration system** for anchoring and automating:

- Troubleshooting & diagnostics
- Contract & service execution
- Public data collection
- Maintenance & support workflows

Its key strengths:

- Verifiability without central control
- Real-time collaboration using local AI agents
- Flexibility to handle structured, semi-structured, and human input

- Minimal setup and low barrier to use—even in legacy environments

Unique Contribution: WRCode introduces a new category of automation: verifiable, contextual, and human-augmented. Users can build their own support systems—scanning a code, adapting the process, capturing relevant evidence, and letting AI assist in root cause analysis, issue resolution, and decision-making. No cloud is required. The user can freely configure the orchestrator according to their own risk policy—running fully local LLMs, using secure cloud-based models, or deploying hybrid setups as needed. This ensures maximum flexibility without proprietary vendor lock-in. The process is **trustworthy by design** and **powerful enough to adapt to any workflow.**

IOTA makes this possible by offering:

- Sustainable, zero-cost anchoring of every workflow element
- Resilience in low-connectivity environments
- A neutral, open framework aligned with public good principles

This project is fully transparent and intended to grow as a community-led standard for trusted, context-aware automation.

WRCode: From Verified Media to Automated Intelligence

How AI Templates and Context-Oriented Orchestration Turn Trusted Data into Action

In a digital landscape where verifying the authenticity of a photo or document is no longer enough, WRCode introduces a new paradigm: **AI-integrated Proof-of-Observation (PoE) sessions**, capable of interpreting trusted data and turning it into actionable insight — securely, privately, and with full lifecycle control.

While existing systems focus on capture-time verification, WRCode is built to operate **after integrity is established** — processing verified media within a programmable automation environment.

What Is a PoE Session?

A **PoE session** is a self-contained, cryptographically anchored bundle that includes:

- One or more **verified input streams** — e.g., media (image, video, audio, speech, text, sensor data, machine logs), documents, device traces
- Optional context (e.g. hardware state, system execution snapshot)
- A **WRCode anchor** that serves as both access token and workflow trigger
- Embedded **AI templates** that define how the input should be analysed, interpreted, validated, and processed

Multiple modalities — image, voice, video, documents, logs — can be grouped into a **single coherent session**, enabling the AI templates to reason across all available inputs together.

Multimodal AI Processing with Private Context

WRCode's orchestration engine allows the analysis of **diverse input types simultaneously**, including:

- Any kind of documents
- Live or recorded voice/video/text/pictures
- Runtime traces or forensic logs of machines or digital environments

The embedded AI templates can access a **locally stored PII Vault or contextual knowledge base** to:

- Cross-reference personal or confidential data without exposing it
- Tailor the logic to domain-specific norms (e.g. regulatory frameworks, company policy, jurisdiction)
- Ensure that all processing remains private and cryptographically auditable

Unlike general-purpose AI models, WRCode enables **user-controlled logic**, ensuring sensitive data never leaves the secure runtime — and that automation reflects the correct operational or legal context.

From Verification to Execution

WRCode does **not** aim to replace secure capture tools or tamper detection services. Instead, it complements them by providing a **modular orchestration layer**, enabling:

- Structured ingestion of media from trusted sources
- Analysis of **multi-source, multimodal inputs** in a single session
- AI-driven interpretation based on **custom templates + local private context**
- Controlled, policy-based routing to designated recipients or storage layers

 *Tamper detection of digital media is a complex scientific discipline requiring specialized models and constantly evolving methods. WRCode makes no claim to replace these systems — instead, its modular, open orchestration architecture allows seamless integration of third-party tamper detection tools into its AI automation templates or validation workflows.*

31 Proof of Execution Packaging: Visualizing and Securing the Full Lifecycle

Each finalized PoE session is securely packaged into a **tamper-evident ZIP archive** that contains:

- The original verified media and context inputs
- Machine-readable metadata
- A signed PDF document including a **WRCode**

- A visual breakdown of the **automation logic and session state**

The PDF outlines (fully customizable and user-defined):

- Where data was routed (e.g. *Google Drive, legal@firm.com, Vault X*)
- What AI logic was applied (e.g. *Template #A203 – contract/video cross-analysis*)
- The current status or result (e.g. *flagged, archived, awaiting co-signature*)

Upon rescanning the WRCode, authorized users can:

- Review the PoE session audit trail
- View results or AI outputs
- Trigger follow-up actions based on the embedded automation logic

This enables **verifiable transparency**, turning static records into **living, auditable workflows**.

✓ Summary: WRCode Enables a New Layer of Meaning and Action

| Feature | Traditional Systems | WRCode |
|---|---------------------|------------|
| Capture-time tamper detection | ✓ | Compatible |
| Runtime attestation | ✓ | Optional |
| Multi-modal session bundling | ✗ | ✓ |
| Embedded AI automation templates | ✗ | ✓ |
| Vault-integrated contextual interpretation | ✗ | ✓ |
| Secure use of local PII/confidential data | ✗ | ✓ |
| Proof of Execution PDF with WRCode | ✗ | ✓ |
| Triggerable lifecycle automation | ✗ | ✓ |
| Zero-trust, private LLM processing | ✗ | ✓ |

 WRCode in One Sentence

WRCode turns verified, multi-source inputs into executable PoE sessions — enabling secure, AI-driven workflows that reason across modalities, preserve privacy, and provide traceable orchestration for real-world decisions.

 Conceptual Approach: Retrofitting Legacy Machines with Edge AI and WRCode-Triggered Automation

Many industrial machines in operation today were never designed with modern AI-driven maintenance in mind. Yet, these machines often provide basic interfaces—control system logs, relay outputs, or operator panels—that can be tapped into for valuable insights.

An emerging concept explores how low-cost edge AI devices, such as the Jetson Orin Nano Super, could act as local anomaly sentinels on legacy machines. These devices are capable of directly communicating with local orchestrators over secure protocols, enabling real-time data exchange, trigger signaling, and feedback loops without reliance on external cloud services. Combined with WRCode's verifiable orchestration framework, this creates a pathway to enhance existing equipment with intelligent, trusted automation triggers—without invasive retrofits or cloud dependence.

▲ Concept Overview: A Passive Anomaly Detection & DeepFix Trigger Loop

The idea centers on a modular edge AI gateway, installed adjacent to the machine. This device would:

1. Continuously monitor available machine data:
 - Control system logs (e.g., from Siemens S7 PLCs)
 - External sensors (vibration, thermal, audio)
 - Visual cues (indicator lights, WRCode scans)
2. Analyze this data in real-time, looking for anomalies, deviations, or fault patterns using lightweight AI agents.
3. Upon detecting an anomaly, trigger a DeepFix session via a WRCode event. The DeepFix process itself would not execute on the edge device but be initiated through a signed, verifiable trigger sent to a higher-level orchestrator.

WRCode Scans as Human-in-the-Loop (HITL) Context Triggers

Beyond automated anomaly detection, WRCode scans can act as a secure, verifiable interaction point between human technicians and the orchestration system. This enables a flexible workflow where human observations and context can be securely injected into the automation loop, enhancing both diagnostic depth and trust.

Workflow Concept:

1. Anomaly Detected → Edge device captures logs, sensor data, visual/auditory cues.
2. WRCode Scan Prompted → The technician scans a WRCode (physical label or digital) that corresponds to the machine, component, or a specific workflow step.
3. Human Context Input:
 - Technician is asked to confirm observable factors (e.g., “Is the valve physically obstructed?”)
 - May be prompted to add notes, photos, or additional measurements
4. Edge LLM Interaction:
 - The technician’s input is directly fed into a small LLM running locally on the Jetson.
 - This allows the LLM to refine its context understanding and provide immediate feedback or suggestions.
5. Secure Packaging & Triggering:
 - The combined data (automated detection + human input) is WRStamped and dispatched as a DeepFix trigger event.
 - A Proof of Execution (PoE) archive is created for audit and follow-up AI loops.

❖ Semantic User Manuals Embedded on Edge Devices

Embedding full user manuals and technical documentation directly onto edge devices allows technicians to access context-relevant sections on-demand, even when no higher-level orchestrator is connected or internet access is unavailable. This semantically searchable knowledge base enables the edge device to interpret real-time events autonomously, ensuring system-specific guidance is always available.

This would allow:

- Technicians to scan a WRCode on the machine to instantly access context-relevant sections of the manual, digitally sign control checks, log cleaning intervals, or record laboratory testing data directly into the system without paper-based documentation, ensuring all actions are verifiable and audit-ready.
- AI agents to query system-specific requirements and constraints when anomalies are detected.
- The Edge LLM to refine its assistance by cross-referencing sensor data and logs with the manual's diagnostic procedures.

Such WRCode-triggered interactions could dynamically guide technicians through troubleshooting steps, maintenance procedures, or safety checks, all tied to the exact machine variant and configuration. Additionally, logging events such as maintenance actions, lab results, and control checks can be automatically dispatched to the local orchestrator, depending on the individual configured AI workflows. This ensures that documentation processes remain automated, yet every entry is verifiably signed and

bound to the operational context without manual paperwork. Additionally, critical maintenance intervals cannot be arbitrarily skipped or ignored. For example, if an oil refill is required, the action can be cryptographically documented at the time of execution, ensuring compliance and traceability without relying on paper logs.

Verifiable Orchestration via WRCode

The key differentiator of this concept lies in the trusted trigger mechanism:

- Each detected anomaly is packaged into a WRStamped event payload, ensuring cryptographic integrity.
- Technician inputs are signed and bound to the machine context through WRCode scans.
- All captured data is bundled into a Proof of Execution (PoE) archive for audit and analysis.

Continuous Improvement through DeepResearch Loops

While the local edge device focuses on immediate anomaly detection and secure trigger generation, more complex diagnostics or pattern recognition across machine fleets could be handled asynchronously. These DeepResearch loops would utilize the PoE archives to refine predictive models, discover root causes, or suggest optimization strategies.

 Potential Advantages of This Approach (Conceptual)

| Aspect | Potential Benefit |
|-----------------------------|--|
| Non-invasive retrofit | Interfaces with existing machine outputs; no need to alter control logic |
| Low cost | Jetson-class devices and sensors provide an affordable entry point |
| Offline-capable | WRCode triggers can operate without persistent internet connectivity |
| Trusted automation | Each orchestration trigger is WRStamped, providing verifiable process chains |
| Human-in-the-loop | Structured technician inputs are securely integrated into workflows |
| Scalable | Can be applied to individual machines or scaled across industrial sites |
| Contextual Knowledge Access | Embedded manuals enable AI-driven, context-specific guidance on-site |

This concept lays the foundation for transparent, accountable, and modular AI-driven maintenance workflows — bridging operational technology (OT) with cryptographic trust layers. Enhancing existing machines with trusted AI-triggered automation doesn't require full-scale modernization. It can start with a modular, verifiable edge node that bridges legacy equipment with WRCode-driven workflows.

Broader Applicability Beyond Industrial Machines

While this concept is illustrated through industrial machine retrofits, the same architecture applies to a wide range of environments where trusted automation and verifiable human interactions are required. Examples include:

- Laboratory equipment logging and compliance workflows
- Medical device maintenance and fault documentation
- Building infrastructure monitoring (HVAC, elevators)
- Secure field service reporting in utilities and critical infrastructure
- Autonomous robots and drones requiring verifiable task execution logs and maintenance documentation, with human-in-the-loop interactions enhanced by granular AI support to assist operators in real-time task validation, context guidance, and secure documentation flows

Any system where context-specific actions, documentation, and verification are essential can benefit from this WRCode-driven, edge-deployed orchestration approach.

Integrating Jetson Nano in Industrial Environments – The WR CODE Approach

Optimando's vision for industrial automation revolves around bridging the inhomogeneous landscape of legacy machines, which often consist of a wide range of proprietary software and hardware components. The goal is to offer maximal flexibility with minimal hardware footprint, ensuring a scalable, GDPR-safe,

and IP-respecting orchestration framework built on open source cryptographic methods. The WRCode concept is inherently open, allowing any vendor, integrator, or service provider to develop and offer solutions under its guidelines. A governance framework ensures WRCode compliance, defining transparent standards for context anchoring, security protocols, and interoperability within the ecosystem. WR CODE Station represents just one implementation example developed by Optimando.ai within this open ecosystem. By reducing hardware dependencies and leveraging existing signal infrastructure, WR CODE significantly lowers integration costs compared to conventional industrial automation upgrades. To achieve this, the Jetson Nano or any other suitable device is not merely treated as a standalone anomaly detector but as a modular node in a holistic orchestration ecosystem.

Technical Integration Concept

Each WR CODE Station is shipped as a compact box, pre-configured for industrial deployment. The station supports modular adaptors:

- Wi-Fi / Ethernet modules for direct network integration.
- Industrial Bus adaptors (e.g., Modbus, CAN, ProfiNet) for multi-device connections.
- Non-invasive signal readers that monitor existing wiring passively, without altering machine circuits.

This design ensures compatibility with:

- Direct machine interfaces (control PCs, PLCs)

- Passive signal interception for non-intrusive monitoring of legacy machines

Device Registration & Context Allocation

Every machine within the environment is assigned a unique WRCode and a human-readable identifier name, enabling intuitive interaction with machine context, memory, and data signals through natural language commands. Machine context and documentation can be allocated by scanning physical documents or by directly emailing the relevant files to the orchestrator, which then maps the content to the correct WRCode-registered machine within the stack. Once the relevant context and documentation have been uploaded and allocated, an operator could simply ask, 'How often do I need to oil machine747?' and receive precise guidance pulled from the machine's assigned context or memory. This feature depends on the successful prior allocation of machine-specific data to ensure accurate and meaningful interactions. This interaction leverages the machine's unique identifier name to directly access relevant maintenance data or operational logs through natural language queries.

- Machine-specific context (manuals, configurations, SOPs)
- Historical logs and allocated memory segments
- Role within the operational hierarchy (visualized in a hierarchical, hash-referenced organigram structure)

Simple machines might not require Jetson Nano-level anomaly detection and can operate with WRCode context alone. The Jetson Nano focuses on localized anomaly detection, while complex orchestration tasks

and live interactions utilize the full computational power of the WR CODE Orchestrator. In advanced scenarios, context and memory can be embedded simultaneously into multiple large language models (LLMs), enabling fine-grained, parallelized reasoning and support for sophisticated automation tasks. This layered and flexible setup ensures seamless adaptation to diverse environments, allowing integration with external sensors and scalable orchestration as needed.

Operator-Centric Orchestration

In most scenarios, a single WR CODE Station is deployed centrally in the operator control room. However, in environments where multiple workstations are cluttered or spatially distributed, the orchestrator can run backend processes that relay data over the network to a central WR CODE hub if applicable. Alternatively, multiple WR CODE Stations can communicate and synchronize directly with each other, adapting flexibly to the specific layout and infrastructure of the environment. It passively monitors:

- Control PC signal outputs (via sniffer modules)
- Network data streams
- User interaction signals (input logs, speech, photos, videos, scanned documents, authorized screen content monitoring for process visualization)

The orchestrator dynamically maps incoming signals to the correct registered WRCode machine in real-time, enabling:

- Anomaly capturing across the entire machine stack

- Contextual allocation of errors, alerts, and maintenance tasks
- Visual feedback through a live browser-based organigram interface

Dynamic WRCode Visualization & DeepFix Sessions

The WRCode organigram functions as a live control map:

- WRCode visual markers dynamically change color codes based on machine state (e.g., maintenance required, critical error, pending lab results).
- Clicking a WRCode triggers an automated DeepFix analysis session, contextualized by its allocated memory and current session data. A WRCode can also be used to sign and mark specific events, such as scheduled maintenance tasks or cleaning intervals. For critical scenarios requiring cryptographically anchored proofs, the signing process can include a fingerprint-confirmed scan, ensuring auditability and formal verification. In less critical cases, a simple click action remains sufficient.
- DeepFix sessions leverage user inputs (text, speech, visual media) utilizing standardized open source AI libraries for input processing and WRCode context to suggest or even execute corrective actions autonomously.

This architecture is agnostic to software ecosystems — the same principle of signal-sniffing and WRCode mapping can be applied to a wide range of data-driven environments, whether it's an industrial control system, an enterprise software suite like SAP, or any specialized platform where data is processed just-in-

time at the point of issue occurrence. This ensures seamless integration across industries, regardless of whether the data originates from physical machine signals or software-driven workflows.



Self-Healing Automation with Customizable AI Templates

By utilizing context-aware memory and modular AI templates, the system evolves into a self-healing, adaptive support framework. DeepFix sessions can be granularly configured to match:

- The criticality of the task
- The scope of intervention (manual approval, semi-automated, fully autonomous)
- The preferred workflow style (operator-supervised or fully automated optimization)

Templates define how AI agents process input streams, ensuring every detected anomaly triggers the most context-appropriate corrective action. WR CODE's modular architecture supports a licensing model tied to WRCode-managed assets, offering scalable deployment options from SME to enterprise environments. All WRCode-compliant systems adhere to transparent public guidelines to ensure compatibility, security, and privacy best practices, without enforcing proprietary lock-ins. WRCode anchoring processes utilize established, publicly auditable hashing standards, ensuring verifiability without introducing proprietary encryption or key exchange mechanisms. The described orchestration workflows are configurable by end-users or third-party integrators and do not prescribe unique AI logic or automation sequences, focusing instead on standardizing context assignment and signal-source allocation.

From Fix to Knowledge: Building Verified Memories with WR Code

On the factory floor, every minute of downtime counts. When a machine misbehaves, experienced technicians often solve the issue after several attempts, testing different adjustments before they land on the fix that works. But here's the problem: the successful solution is often not written down, or it gets forgotten after some time. Proper documentation takes time, and in a production environment time is a scarce resource. And when experienced staff leave the company, their knowledge leaves with them.

WR Code changes this dynamic. Every WR Code-registered asset or workflow can be extended with its own knowledge memory, a verified record of solved issues. Instead of starting from zero when a problem reoccurs, technicians can query the system and find a validated fix—complete with photos, screenshots, or diagrams—right at the machine.

And the effort to capture this knowledge doesn't need to be manual. With DeepFix and multi-agent AI orchestration, documentation can be created semi-automatically:

- The AI assists in recording the problem description, attempted fixes, and the final solution.
- Evidence such as measurements, logs, or photos can be attached directly during the troubleshooting process.
- The system validates and organizes this information before adding it to the knowledgebase.

The real power comes during retrieval. Semantic search across validated entries means teams don't have to remember exact keywords. They can simply describe the issue in their own words, and the system surfaces the most relevant, previously solved cases. Combined with the stored documentation, this provides maximum support for recurring issues—whether on a packaging line, a PLC configuration, or an SSL setup in factory IT.

Beyond retrieval, the system can also count and statistically analyze issues over time. If the same type of error keeps reappearing, automated suggestions can be generated to address the root cause—turning the knowledgebase into not just a memory, but also a driver for continuous improvement.

And the principle doesn't stop at manufacturing. The same memory model applies to IT systems, logistics workflows, or any WR Code-registered process: solve once, document semi-automatically with AI support, and reuse the knowledge many times over—even when the original expert is no longer part of the company.

In practice, this means fewer repeated mistakes, faster onboarding for new staff, actionable insights into recurring problems, and a gradual shift from reactive firefighting to a continuously growing base of verified operational knowledge. With WR Code, every fix becomes an asset.

Automatic Context Generation Through OCR and Multi-Context Analysis

A core challenge in automation and orchestration is the creation of structured context data. While WRCode templates can be supplied directly by manufacturers or system operators, many real-world scenarios lack ready-made structured input. Automatic context generation helps bridge this gap.

Modern OCR and AI-based parsing techniques allow unstructured sources such as labels, receipts, or manuals to be transformed into structured WRCode snapshots. For example, photographing a food package enables automatic extraction of nutrition facts, ingredient lists, and weight declarations. These data points can then be normalized and verified against public databases or user corrections. Once verified, the context is embedded into a vector database, where it can be semantically searched and linked to related information. From there, it becomes directly usable for downstream automation processes such as refined multi-AI-agent analysis.

Beyond pure extraction, the captured data can be refined and enriched by large language models. An LLM can correct OCR misreadings, harmonize units, infer missing micronutrients from typical values, and categorize ingredients into broader dietary classes. This enrichment step transforms a basic snapshot into a more complete and semantically usable context.

The real strength emerges when multiple context classes overlap. A nutrition log by itself only shows intake. Combined with personal activity data, blood values, or occupational context, the orchestrator can detect recurring deficiencies and provide targeted suggestions. For example, a sportsperson may show insufficient magnesium or iron intake, highlighting the need for mineral-rich foods. An indoor worker may face a vitamin D deficiency due to limited sunlight exposure, making supplementation or dietary

adjustments relevant. Similarly, vegetarians and vegans often require special attention to vitamin B12 intake, which is difficult to obtain from plant sources alone.

The same principle applies in industrial settings: OCR of maintenance logs, paired with machine telemetry and shift schedules, allows automated correlation that reveals patterns otherwise overlooked.

This approach turns context creation into a semi-automatic process. Users are freed from exhaustive manual input, while the orchestrator gains access to a growing, verified dataset that reflects real conditions. Over time, overlapping contexts produce a knowledge layer where recurring issues are automatically documented, searchable, and statistically analyzed. In practice, this reduces knowledge loss when team members leave, accelerates troubleshooting, and enables continuous optimization.

By anchoring automatically generated contexts with WRCode, embedding them into a vector database, and linking them into the orchestrator, everyday processes—from food choices to industrial maintenance—can be analyzed holistically. The result is not a black-box automation, but a user-guided automatic context extraction process: users provide minimal guidance, OCR delivers raw structure, LLMs refine and enhance the data, and multi-AI-agent orchestration unlocks downstream value.

OptiScan: Efficiency Analysis & Possible Productivity Gain Identification for Complex AI Orchestration

In multi-AI-agent environments, performance bottlenecks are often hidden — not in faulty code or slow hardware, but in missing knowledge, bad habits, outdated context, or inefficient workflow logic. OptiScan is built to uncover these blind spots.

It's more than a performance audit — it's the Orchestrator's full-spectrum analysis tool, designed to identify where efficiency is being lost and how it can be recovered.

What Is OptiScan?

OptiScan is a targeted optimization process within the multi-AI-agent Orchestrator.

Its mission:

- Scan all available operational data — requiring the user to set a defined time span for analysis — including context, memory stacks, and AI templates.
- Identify knowledge gaps and missed optimization opportunities.
- Suggest precise, actionable improvements to increase overall workflow efficiency.
- Package the findings into a PDF report containing a WRCode for easy review, sharing, and discussion.

While anyone can use it, OptiScan is specifically designed for complex environments with multiple WRCode-registered devices and machines, where small changes can create significant performance gains. The OptiScan session itself can be fully customized to fit the operational and analytical needs of each case. Users can also add additional data — such as factory layouts, ground plans, photos, videos, number of employees, or other operational parameters. The more data provided, the higher the chance of uncovering valuable optimization opportunities.

How OptiScan Works

Data Scope Definition

User or system sets the analysis time window — from hours to months — ensuring both historical and recent patterns are covered.

Comprehensive OptiScan Processes:

1. **Context Stack** — All embedded data, whether publicly fetched or locally enhanced, combined with operational conditions, orchestration rules, and environment variables. This includes sensor data, machine states, and real-time environmental inputs.
2. **Memory Stack** — Logs, learned behaviors, and historical workflows. Extended to incorporate:
3. **Process Handling Recordings** from AR glasses worn by employees, enabling step-by-step replay and analysis of real-world procedures.
4. **DeepFix Session History**, providing insight into past troubleshooting steps and outcomes to spot recurring issues and pattern-based failures.
5. **AI Templates** — Logic blueprints for each WRCode-registered device or AI agent. Templates dynamically adapt based on context and memory insights, incorporating visual and operational data to recommend optimizations, suggest alternative processes, or prevent recurring errors.
6. **Gap & Potential Identification**
Cross-references what should happen vs. what actually happens to find inefficiencies and blind spots.

7. Optimization Proposal

Produces a prioritized list of changes — from knowledge injections to workflow adjustments — aimed at measurable efficiency gains.

8. Result Packaging

All results are compiled into a PDF report containing a WRCode that links directly to the session's findings for quick access. The output can be multimodal, including text suggestions, charts, mock-ups, visualizations, content recommendations, or any other elements that may support decision-making. Results are presented in a predefined grid format, often optimized for multi-screen display setups.

Why It Matters for Business Processes

- **System-Wide Awareness** — Analyses all relevant layers instead of looking at isolated devices.
- **Business-Centric Improvements** — Designed to enhance workflows in enterprise and industrial processes, where coordination between multiple agents is mission-critical.
- **Adaptable to Any Scale** — From small operations to large industrial setups.
- **Local-First Security** — All analysis runs on-premise unless otherwise configured, protecting sensitive process data.

Example: Manufacturing Workflow Optimization

A production facility runs dozens of WRCode-enabled machines, each managed by dedicated AI agents. Over the past six months, inefficiencies have crept in:

- Machine calibration tasks are triggered too frequently.
- Certain AI agents re-check data already validated elsewhere.
- Peak-hour workflows stall because one machine's AI lacks updated shift schedule context.

OptiScan scans all data from the last 180 days, identifying:

- A 15% potential throughput increase by aligning calibration with real wear metrics.
- A reduction in redundant validation steps through better memory synchronization.
- An optimized task distribution plan during peak hours.
- Suggestions for additional or alternative sensors, machines, or equipment to further enhance productivity.

The findings are packaged into a PDF with a WRCode, enabling managers to open the session results instantly for review and team discussion.

DeepFix vs. OptiScan

| Feature | DeepFix | OptiScan |
|---------------|---|--|
| Primary Goal | Detect & fix specific issues | Detect knowledge gaps & optimization opportunities |
| Scope | Fault repair | System-wide efficiency analysis |
| Data Analyzed | All available data — including fault logs and error states — everything that might be helpful, but with the focus on fixing the issue | All available data — including context, memory, AI templates, full historical data, and any optional operational data — everything that might be helpful |
| Output | Fix suggestions + Deepfix report | Efficiency improvement plan + OptiScan report |
| Ideal Use | Resolve malfunctions quickly | Improve workflows & processes |

OptiScan transforms orchestration from reactive troubleshooting to proactive, data-driven optimization — ensuring that every device, agent, and workflow in a WRCode-enabled ecosystem works at its maximum potential.

WRCode — Turning Any Product into a Smart, Interactive Experience

WRCode can provide products with an interactive layer. By combining registration with context, memory, and embedded metadata, objects function as accessible, intelligent assistants.

This approach removes the need to search through manuals: a single scan allows direct interaction via voice command. Printers, routers, household devices, or packaged goods can be extended into a multi-AI automation environment.

Example – Printer:

- “Order new paper and cartridges from Amazon.”
- “Do I need special photo paper for this printer?”
- “How do I scan documents and send them to my email?”

Restaurants may let guests scan a table code and speak to explore menus, confirm dietary details, and place orders. Hospitals could equip medical devices with WRCode, enabling staff to request protocols or maintenance steps hands-free. Retailers may attach WRCode to electronics so customers and staff can access troubleshooting, manuals, or warranty information instantly. Looking forward, cars, industrial machines, or even urban infrastructure could offer context-aware voice interaction through WRCode. With WRScan, multi-agent workflows, semantic retrieval, and personal notes become available across domains — from consumer goods to healthcare and smart cities. The process is frictionless: no setup required, but fully extensible for any kind of downstream AI automation. WRCode doesn’t just connect products to the digital world — it makes them smart.

Concept: Voice-Interactive Menus and Unified Ordering Framework

The WR-Code standard can be applied to hospitality and retail environments to streamline menu browsing, ordering, and payment processes. In this concept, each table, counter, or pickup area includes a WR-Code that customers scan using a smartphone or AR glasses. Once scanned, the system identifies the venue and provides access to the corresponding digital menu or product catalog.

Publishers can set this up with minimal effort. A standard PDF containing item descriptions, images, prices, and regulatory details can be uploaded, and the system extracts and normalizes this content into a structured format. The extracted text is embedded into a vector database, allowing customers to interact conversationally. They can ask questions such as “*Which dishes are lactose-free?*” or “*What is recommended for lunch?*”, and receive precise responses derived directly from the restaurant’s own data.

WR-Codes printed by the publisher may optionally include table numbers or location identifiers, allowing the system to automatically assign each order to the correct table or seating position. This enables staff to receive clear, structured order information without requiring verbal clarification or manual entry.

Ordering and payment workflows remain flexible. Customers may pay immediately, pay digitally after the meal, or settle in cash. The aim is to minimize procedural delays and allow guests to simply enjoy the experience. For instance, a customer may pre-order and optionally prepay when reserving a table online by scanning the WR-Code on the restaurant’s website. The meal can then be prepared to coincide with the guest’s arrival, reducing waiting times and peak bottlenecks.

Privacy-preserving on-device geofencing ensures that restaurants may limit ordering to on-site guests without transmitting location data externally. If remote ordering or delivery is intentionally permitted, the system may request address and delivery preferences during checkout. No personal, location, or behavioral

data leaves the device without explicit user consent, including at the moment of scanning. Scanning a WR-Code alone does not transmit data.

Larger venues may optionally integrate stock or availability data to reflect real-time conditions. The system provides a consistent method for linking such data sources, following the same core principle: content is ingested (e.g., via OCR or connectors), normalized, and embedded into the WR-Code's structured data format. Processing can remain fully on-premise if desired, and data only leaves the controlled environment when an order or payment is intentionally submitted.

All incoming orders appear in a unified real-time interface for staff—regardless of whether they were placed remotely, in advance, or on-site. This reduces operational overhead and allows staff to focus on service and fulfillment rather than administrative coordination.

This framework is presented as a conceptual model, though all components are technically feasible with existing technologies. The purpose is to illustrate how WR-Code can reduce workflow friction for both customers and establishments, enabling smoother, more efficient experiences with minimal setup effort.

Multimodal Context Embedding — A New Dimension of Interaction

Most large language models today primarily embed and process text. However, a new field is rapidly emerging: multimodal context embedding, where models also integrate images, video, 3D animations, and other data streams. WRCode is designed to be ready for this evolution.

Publishers and users can enrich WRScans with multimodal assets:

- Installation guides enhanced with videos, images, or AR overlays
- 3D animations that demonstrate assembly steps (e.g., building furniture)
- Training snippets for operating machinery or consumer devices
- AR/AI Glasses integration, where users can ask:

"I don't understand where to put this piece."

and instantly see a video overlay or contextual animation guiding them.

Because WRCode already embeds context, memory, and orchestration logic by design, it can seamlessly extend into this multimodal domain. The optimization layer is capable of supporting not only explicit commands but also proactive, context-driven guidance — ensuring that products can "teach themselves" to their users.

In short: WRCode gives every registered product customizable brains — ready for today's text-based assistants and tomorrow's multimodal AI.

LetMeGiraffeThatForYou: Visual Question Capture and Automated AI Orchestration Trigger in WR CODE

As part of the **WR CODE** AI orchestration framework, we introduce "**LetMeGiraffeThatForYou**"—a decentralized feature that allows users to instantly capture any part of their digital environment and trigger an AI-powered research process with zero manual input, simply by clicking a button.

How It Works (with Privacy-First Manual Control):

1. A floating capture button is available inside the browser, workspace, or any **WR CODE**-integrated environment.
2. When clicked:
 - The user can manually select an area of the screen or capture the full window—for example, a chat question, social media post, or any on-screen content.
3. The captured input—whether screenshot, screencast, or structured data—is passed into **WR CODE**'s master orchestrator, where:
 - AI automatically detects the question or intent from the selected content.
 - A predefined AI research workflow is triggered, running through local models, cloud services, or both, depending on user preferences.
4. The system then generates:
 - A context-aware answer or insight.
 - A branded, share-ready PDF, visual snapshot or a video capture, automatically copied to the clipboard for instant pasting into any chat, post, or document.

 **Optional Manual Trigger for Maximum Privacy:**

- **WR CODE** is designed with a continuous optimization layer that can automatically monitor digital activities (e.g., tab context, page content) to suggest proactive AI assistance when enabled.
- For users or organizations who prefer strict control or offline operation, this continuous layer can be toggled off.
- Users explicitly choose when and what to capture (screenshot, screencast, text).
- No background monitoring or context capture takes place unless initiated by the user.

The process is triggered only when explicitly requested for selected input types, ensuring that users retain full control over when and how AI-driven analyses are initiated. This is particularly relevant for users who choose to toggle off the automatic optimization layer within **WR CODE**: in such cases,

LetMeGiraffeThatForYou allows users to manually capture input and selectively trigger AI research only when needed. In team discussions, online chats, or collaborative workspaces where questions naturally arise, **LetMeGiraffeThatForYou** enables users to provide instant, AI-assisted answers through a simple click-and-share mechanism—making knowledge sharing effortless, efficient, and privacy-friendly.

Key Features of LetMeGiraffeThatForYou

-  **User-Defined Capture Area:**

Users can select the **exact portion of their screen** to capture—whether it's a single question, an entire chat window, or a complex discussion thread. Flexible for **full screens, snippets, or specific visual content**.

-  **Automatic Intent Recognition:**

The system's AI **autonomously detects the last question, request, or informational need** from the captured content—without requiring manual prompting (depending on the selected workflow template).

-  **Preconfigured AI Research Workflows:**

Upon capture, the system triggers a **predefined WR CODE orchestration**, executing **parallel AI research agents** to synthesize accurate, context-aware answers. Research can run **fully locally** or leverage **hybrid cloud models**, as configured by the user.

-  **Privacy-First & Decentralized by Design:**

All processing can operate **100% locally** with **no centralized data storage or tracking**. Alternatively, users can **opt-in to cloud AI** for enhanced capabilities. Full control remains with the user or organization at all times.

-  **Instant Visuals for Viral Sharing:**

LetMeGiraffeThatForYou creates **automatically branded, shareable visual outputs (PDFs, screenshots, video snippets, or even audio messages)** that are copied automatically into the clipboard and can be pasted directly into chats, posts, or documents—enabling seamless knowledge-sharing.

Strategic Position: The domain LetMeGiraffeThatForYou.com has been secured to protect the naming rights of this concept, which refers to a built-in feature within the WR CODE orchestration framework. The

underlying concept, technical design, and process constitute original intellectual property (IP) within the WR CODE ecosystem. By combining customizable visual capture, automated question detection, and orchestrated AI research, LetMeGiraffeThatForYou introduces a playful yet powerful way to simplify digital interactions, enhance knowledge sharing, and promote AI adoption—without reliance on centralized infrastructure, if desired.

LetMeGiraffeThatForYou is designed for quick, straightforward results where no deepfix sessions, Proof-of-Operation (PoE) validation, or in-depth analysis workflows are required. It offers a lightweight yet powerful method to instantly share answers and insights in online scenarios like team meetings, project collaborations, or private chats on social media. It focuses on speed, simplicity, and ease of use, making it the perfect tool for on-the-fly research responses and content sharing without the overhead of complex automation processes.

Agents in WR CODE

Agents in WR CODE are modular units organized into four sections: Listener, Context, Reasoning, and Execution. This layered structure provides a flexible “construction kit” for automation that adapts to both simple personal tasks and complex multi-agent workflows.

1. Listener

Listeners are the entry points for events and can operate in several ways:

- Active Listener – reacts to explicit triggers such as:

- **Command identifiers** – which can be defined for *anything* (e.g. #business mail5, #calendar9, #tax17, #research42).
- **Uploaded files**
- **Pinned screen areas with tags**
- **Voice memos, video clips, or sensor data** — captured on mobile devices, Smartglasses, or robots — each embedding one or more command identifiers.
- **Passive Listener** – observes context and detects patterns in the background.
- **Fine-tuned Local Models** – provide built-in pattern recognition for recurring tasks (e.g. email drafting, scheduling, document analysis). These patterns can themselves act as triggers for automation tasks, depending on user-defined settings and priorities.

Practical example (personal use): A user records short voice memos or videos during the day — on a train, with Smartglasses while walking, or via a mobile phone — embedding identifiers like #mail5, #calendar9, or #compare. Once synced at home or on a workstation, the system transcribes these inputs and automatically activates the corresponding agents: drafting emails, preparing calendar entries, running tax workflows, or conducting a product comparison based on the recorded material.

Practical example (industrial use): A robot packing parcels detects command identifiers (e.g. QR codes, visual markers, or sensor triggers). These identifiers translate into backend automation tasks such as inventory updates, shipping label generation, or quality logging. A backend operator can either review and approve tasks in real time or collect them into a backlog to run later in a structured session. This ensures

that even operator breaks or shift changes don't disrupt the workflow, while keeping a human-in-the-loop for oversight.

While some note-taking and automation tools capture parts of this idea, the concept of embedding identifiers into multimodal inputs (voice, video, or sensor data) to orchestrate multi-agent workflows across personal devices, Smartglasses, or robots is not commonly seen in other systems and may represent a fresh approach — though we describe it cautiously as a possible innovation.

2. Context

- **Global Context** – shared across the orchestrator and usable across multiple sessions, ensuring continuity of knowledge and workflows.
- **Agent-Specific Context** – scoped to an individual agent, enabling highly tailored memory or instructions.

3. Reasoning

The reasoning section interprets listener input together with context and determines what should happen next.

- Applies templates, optimization rules, and orchestration logic.
- Can remain lightweight (default rules) or be extended with deeply customized agent logic.

4. Execution

The execution section carries out the requested tasks — drafting business emails, generating calendar entries, preparing tax calculations, running analytics, or triggering downstream integrations.

- Execution can run immediately (e.g. on a mobile instance or during robot operation) with a human in the loop if required, or later in a workstation session for review and oversight.

System Agents

WR CODE also includes system agents that orchestrate and secure the overall environment:

- Input Coordinator – a meta-listener monitoring all listeners and integrating fine-tuned LLM pattern recognition.
- Output Coordinator – manages formatting, merging, and delivery of results.
- Specialized Coordinators – e.g. anomaly detection, DeepFix (error correction), or OptiScan (efficiency analysis).

System agents are editable by design, offering maximum flexibility for users who want to adapt even the core orchestration logic to their compliance requirements, workflows, or optimization needs.

WR CODE also includes system agents that orchestrate and secure the overall environment:

- Input Coordinator – a meta-listener monitoring all listeners and integrating fine-tuned LLM pattern recognition.

- **Output Coordinator** – manages formatting, merging, and delivery of results.
- **Specialized Coordinators** – e.g. anomaly detection, DeepFix (error correction), or OptiScan (efficiency analysis).

System agents are editable by design, offering maximum flexibility for users who want to adapt even the core orchestration logic to their compliance requirements, workflows, or optimization needs.



Glassview: Making AI Coding Tools Transparent Through Multi-Agent Analysis and Human-in-the-Loop Refinement

Modern AI coding tools such as Cursor, Claude, and GPT-based systems have become essential components in contemporary software engineering workflows. Their ability to generate meaningful insights, propose architectural improvements, and assist with complex reasoning tasks has significantly accelerated software development. Glassview, built within the broader WR CODE orchestration environment, is designed to amplify these capabilities by adding a transparent, developer-controlled reasoning layer that coordinates multi-agent analysis and collaborative refinement.

Glassview does not replace AI coding tools, nor does it attempt to replicate their core strengths. Instead, it serves as a transparent orchestration and analysis framework built around them—enhancing their usefulness, surfacing their internal reasoning, and structuring the complex thinking processes that occur before code is written or modified.

1. Developer-Defined Templates: The Foundation of Structured Intelligence

Unlike traditional coding assistants that respond directly to user prompts, Glassview begins with developer-defined analysis templates. These templates represent structured workflows that define:

- what the AI should analyze,
- which aspects of a problem should be scrutinized,
- how hypotheses should be evaluated,

- what patterns or metrics matter,
- and which reasoning modes should be applied.

These templates act as customizable cognitive blueprints. They allow teams to encode their own engineering principles, debugging strategies, architectural heuristics, or domain-specific knowledge into repeatable AI thinking processes. This gives teams full control over *how* the AI reasons—an essential requirement for complex engineering contexts.

2. Multi-Agent AI Brainstorming Based on Developer Logic

Once a template is defined, Glassview executes a multi-agent AI brainstorming session based on its structure. Instead of one model producing a single answer, several different AI agents—each with distinct capabilities or reasoning styles—contribute insights.

These agents may:

- explore different hypotheses,
- generate alternative interpretations,
- propose potential causes of observed behavior,
- compare inconsistent reasoning paths,
- identify missing information,

- or suggest strategic debugging directions.

Because the agents operate within developer-defined templates, the brainstorming is structured, predictable, and aligned with the team's methodology.

The result is a diverse set of AI-generated ideas, each grounded in the same analytical framework.

3. Extracting and Ranking the Best AI Findings

Glassview automatically analyzes the output of all participating AI agents.

It filters and ranks the findings using several criteria:

- clarity and coherence
- feasibility
- architectural alignment
- consistency with known constraints
- explanatory strength
- potential impact

These evaluations are not arbitrary.

Glassview uses multiple LLMs—each scoring ideas through different reasoning filters—to ensure balanced and comprehensive assessments.

The top-ranked ideas form a curated set of high-value insights, which are then forwarded to the team.

4. Team-Based Collaborative Refinement

The selected AI findings appear in the Glassview Stream, a shared, real-time feed visible to all team members. This becomes the nucleus of collaborative reasoning.

Teams can:

- refine ideas**
- merge complementary insights**
- critique assumptions**
- extend explanations**
- challenge weaknesses**
- compare alternatives**
- and evolve proposals into mature, actionable strategies**

Glassview integrates action buttons next to each idea, enabling:

- *Iterate*
- *Extend*
- *Critique*
- *Generate Test Scenario*
- *Compare with Past Issues*
- *Highlight Dependencies*

This transforms brainstorming into an interactive, cyclical reasoning loop.

Developers remain in control, while AI provides structured support and diverse perspectives.

5. Transparent Reasoning: Turning AI into an Explainable Partner

A key innovation of Glassview is its dedication to transparency.

It reveals:

- each agent's reasoning path,
- step-by-step logic,

- points of contradiction or uncertainty,
- why certain ideas were ranked highly,
- how findings relate to developer templates,
- and how ideas evolved across iterations.

Instead of black-box answers, developers see explainable, inspectable reasoning chains.

This allows teams to:

- trust AI recommendations more deeply,
- detect flawed assumptions,
- leverage diverse reasoning styles,
- learn from AI-generated insights,
- and incorporate AI reasoning directly into team discussions.

Glassview enhances—not replaces—the intelligence already present in modern AI coding tools.

6. Real-Time Mini-Apps Supporting Idea Exploration

When the AI or a team's reasoning workflow requires additional clarity, Glassview can generate micro-interfaces (mini-apps) tailored to the problem at hand. These may include:

- visual debugging overlays,
- dependency or event-flow diagrams,
- sandboxed test environments,
- hypothesis-verification tools,
- interactive comparison panels,
- or complexity-analysis widgets.

These mini-apps help developers validate ideas, explore consequences, or understand complex reasoning steps interactively.

They act as localized, real-time assistants that support the brainstorming process.

7. Learning-by-Doing: Embedded Pedagogical Support

Glassview transforms every analysis session into a learning opportunity.

Because developers can inspect detailed reasoning processes, explanations, critiques, and template-driven workflows, the platform becomes a continuous learning environment.

For beginners:

- Glassview clarifies language constructs, logic flows, and common pitfalls.

For intermediate developers:

- it highlights architectural tradeoffs, performance considerations, and pattern recognition.

For experts:

- it reveals complex system interactions and uncovers subtle insights.

Glassview thus unifies education, collaboration, and reasoning, embedding knowledge transfer directly into the development workflow.

8. A New Model for AI-Augmented Engineering

Glassview represents a shift from prompt-based interactions toward template-driven, multi-agent, team-centered reasoning ecosystems. It enhances the already extraordinary capabilities of AI coding tools by adding:

- transparency,

- **structure,**
- **collaboration,**
- **iterative refinement,**
- **and developer-defined cognitive workflows.**

Rather than automating developers out of the process, Glassview amplifies human expertise by turning AI-generated insights into shared, explainable, and actionable reasoning artifacts.

Teams benefit from:

- **deeper understanding,**
- **faster convergence on complex issues,**
- **more robust decision-making,**
- **and transparent collaboration.**

Combined with WR CODE, Glassview marks a step toward a new paradigm in software engineering—one where AI tools remain powerful collaborators, and human teams remain fully empowered through structured, transparent, and explainable reasoning loops.

[**Introducing TimesDesk: A Partially Decentralized System for AI-Driven Asset Comparison, WR Code Templates, and Realtime Crowd-Enhanced Scoring**](#)

TimesDesk is a partially decentralized framework for exploring financial products and other assets through private, local AI analysis combined with an optional community-driven scoring layer. It connects a locally running AI orchestrator with the augmented overlay of WR Code, allowing users to define customized analytical workflows while maintaining full control over the data processed on their own devices.

At its core, TimesDesk offers a flexible asset comparison engine. When a user searches for an asset—whether a stock, cryptocurrency, commodity, index, or another type of instrument—the system produces a broad set of potential comparison pairs (“swap opportunities”). Each pair is evaluated through a locally executed scoring model that accounts for factors such as historical performance, volatility, and contextual relationships. All reasoning occurs on the user’s device through the orchestrator, ensuring a privacy-preserving analysis pipeline.

Customizable analysis templates are enabled by the WR Code augmented overlay. Within this overlay, users can define their own evaluation logic, criteria, and procedural steps that guide the orchestrator’s reasoning. These templates remain fully user-owned. TimesDesk then consumes the anonymized outputs of these WR Code-based workflows, optionally generating template-quality scores to help users refine or standardize their analytical style.

Users who opt in may share anonymized analysis results—including comparison outputs, scoring metadata, and template-driven evaluations—into a collective dataset. Contributors receive collaboration credits, which unlock access to deeper aggregated scoring, extended comparison options, and more refined ranking models. The combination of these shared results forms a community-generated insight layer built on principles similar to swarm intelligence, without requiring the disclosure of personal data.

A central component of the system is the ticker-style realtime feed. As users perform comparisons through their WR Code overlay and orchestrator, anonymized swap analytics are streamed back into TimesDesk as live, rolling updates. This feed highlights emerging relationships across financial products and other assets, showing trending comparisons, changes in user focus, and frequently analyzed pairs based on aggregated activity.

The architecture is intentionally lightweight:

- asset data is retrieved from public or licensed APIs,
- the AI orchestrator performs all private, template-based reasoning locally,
- a minimal backend stores anonymized contributions and collaboration credits, and
- WebSockets deliver the realtime ticker and aggregated scores to the frontend.

Together, these components establish TimesDesk as a partially decentralized, privacy-focused analytical environment. By combining local AI processing, customizable WR Code templates, and a collaborative scoring layer, TimesDesk provides a structured approach to understanding relationships across financial products and other assets while preserving user autonomy and data control.

Decentralization Roadmap for WR CODE's WRCode Architecture

Introduction The WR CODE project, including its WRCode protocol, is built on one fundamental principle: **trust must never rely on a single party**. In a world where digital infrastructure is increasingly fragile and central points of failure are vulnerable to attack, manipulation, or censorship, WR CODE proposes a radically different approach. WRCode is being designed from the ground up to be verifiable, tamper-proof, and independent of any one actor — including its creator.

This article outlines a conceptual decentralization roadmap for the WRCode ecosystem, ensuring that its integrity, usability, and business potential can remain intact for decades to come.

Why Decentralization Matters

- **Resilience:** Centralized systems are attractive targets. A single point of failure compromises trust for everyone.
- **Transparency:** Decentralized, verifiable protocols allow users to audit and validate independently.
- **Longevity:** Protocols survive long-term only when they can evolve beyond their original operator.
- **Selective Neutrality:** While WRCode's verification infrastructure is open for anyone to run and self-host, participation in the official trust network (e.g., verified publisher onboarding, WRStamp issuance, public indexing) is governed under clear policy and subscription terms. This ensures legal and operational integrity while preserving transparency at the protocol level.

The Roadmap: Trustless by Design, Distributed by Strategy

Phase 1: Anchored Trust, Centralized Coordination (*Initial Concept Phase*)

- WRStamp anchoring via Solana + OpenTimestamps (planned)
- Template, manifest and context hosting via WRCode.org (centralized starting point)
- Publisher verification and orchestration hosted on wrcode.org infrastructure
- Monetization through verified publisher accounts, WRStamp APIs, and trust-based onboarding services

In this early conceptual phase, critical functions are envisioned to be cryptographically verifiable, even if coordination remains centralized.

Phase 2: Distributed Verification and Self-Hosting (*Planned Mid-Term Goal*)

- Release of wrcode-node: a lightweight node for mirroring WRStamp anchors and verifying WRTemplates locally
- Publisher keypair system introduced to decentralize license trust (signatures replace centralized approval), while still enabling monetization through subscription-based identity verification
- Caching and validation infrastructure for Merkle-root anchoring of large template batches
- Decentralized participants can verify, anchor, and execute templates independently — but will not be onboarded into the trusted publisher network without registration through Optimando.ai

This phase proposes that anyone can host, verify, and trust WRCode artifacts without relying on Optimando's infrastructure, though premium trust layers remain gated.

Phase 3: Protocol Sovereignty and Governance (*Long-Term Vision*)

- Governance of WRCode protocol envisioned through a neutral protocol council or distributed policy board, composed of legal, technical, and stakeholder representatives
- Template indexing and WRStamp anchors mirrored across IOTA, Bitcoin, Solana, Optimism, IPFS, and DNS TXT chains
- Community-hosted WRRegistry instances for federated search and trust models
- Orchestrators able to resolve and validate WRCode payloads from any source without privacy compromise
- Optional **smart-contract-based rule enforcement**, governed by multisig legal policy boards or decentralized autonomous oversight — enabling legally verifiable bans, auto-revocation, or audit-triggering behavior encoded into protocol logic
- All WRTemplates are instruction-based, not executable code. Each template is double-guarded through:
 - Signed provenance and cryptographic stamping

- Mandatory AI-based filter templates that evaluate logic before execution

This ensures that even deceptive AI instructions are intercepted at runtime, reducing the risk of orchestrator misuse without relying solely on static audits.

Importantly, WRCode decentralization ensures no single actor can compromise the protocol, while still allowing Optimando.ai to retain control over monetized onboarding and service offerings — preserving legal and operational safeguards for the protocol's founder.

Instruction Verification Layer (AI Filter Chains)

To further mitigate misuse of non-code instructions, WRCode introduces a layered instruction validation mechanism:

- **AI Checker Templates:** Enforced logic filter modules that run before any WRTemplate is allowed to execute. These filters analyze instruction content and context using lightweight LLMs or symbolic rule systems.
- **Policy Chaining:** Orchestrator policy can define mandatory filters. For example, a WRTemplate may only execute if passed through WRFilter-AI-LegalCheck and WRFilter-FormSanity.
- **Signed Filter Templates:** Filter AIs themselves are WRStamped and versioned, ensuring consistency and traceability.
- **Bypass Detection:** If a malicious template bypasses filters or manipulates the instruction payload, the orchestrator flags the WRStamp as compromised locally and may submit a signed incident report.

Revocation from the trusted registry requires verification through governed policy (e.g. multisig protocol board or encoded rule logic), ensuring no single actor can unilaterally alter protocol trust status.

Even though WRTemplates contain no executable code, this layer ensures that intent and behavior remain under strict control — enforced by both protocol logic and AI interpretation.

At this stage, WRCode would evolve into a neutral public protocol. The creator would no longer be a bottleneck or a target — but may remain a critical innovator and economic operator.

Conclusion: Long-Term Trust Through Transparent Design

The future of WRCode and the WR CODE automation ecosystem rests on a clear mission: **no single point of failure, no unverifiable logic, and no blind trust required.**

Decentralization isn't a feature — it's a survival strategy. By embedding this principle into its architectural vision from the start, WRCode aims to ensure a future where automation is not just powerful, but accountable, resilient, and trustless by design.

A Vision of Orchestrated Robotics Built on WRCode

In 10 to 20 years, robots will be cheaper and more reliable than human labor, operating with near human-level fault tolerance. This future will not come from trial-and-error deployment but from millions of orchestrated training cycles in digital twin environments

such as Omniverse. Entire factories, logistics hubs, and service fleets will already have been optimized virtually, so that when robots arrive in the physical world, they can work almost failure-free from day one.

Just as today's AI ecosystem balances general-purpose LLMs with specialized agents, tomorrow's robots will follow a similar pattern. Some will be versatile generalists, others fine-tuned specialists. The crucial enabler is orchestration: ensuring that the right robot is activated at the right time, equipped with the proper skills, context, and data. Orchestration is what turns isolated machines into a synchronized, adaptive workforce.

Here, WRCode and WR CODE provide the backbone. Every robot, whether simulated or physical, must be registered, verified, and auditable within the WRCode system. A robot that does not meet these conditions will simply not run. This rule enforces tamper-proof decision trails and immutable orchestration logs — enabling every action to be traced, verified, and, if necessary, halted. Inbuilt kill-switch mechanisms are not a theoretical concept but a practical safeguard against sabotage, malfunction, or systemic failures.

Failures, when they occur, are managed through DeepFix sessions, where the orchestrator replays anomalies in simulation until robust solutions are validated and redeployed. Beyond fault recovery, OptiScan sessions continuously analyze orchestration logs, embedded data,

and context stacks to uncover hidden inefficiencies and propose system-wide optimizations.

This creates a powerful self-reinforcing loop:

- Robots grow cheaper, faster, and more capable.
- Orchestration frameworks ensure dynamic coordination of generalist and specialist machines.
- LLMs enhance reasoning across teams of agents.
- Computation enables trillions of orchestrated cycles in simulation, where robots are trained virtually before being applied to real-world floorplans. Once deployed, they can be further fine-tuned by employees. WR Code provides the interaction layer, ensuring orchestration follows a human-in-the-loop approach.
- WRCode anchoring establishes trust, compliance, and auditability.
- DeepFix sessions turn failures into reinforced improvements.
- OptiScan sessions continuously optimize performance beyond human foresight.

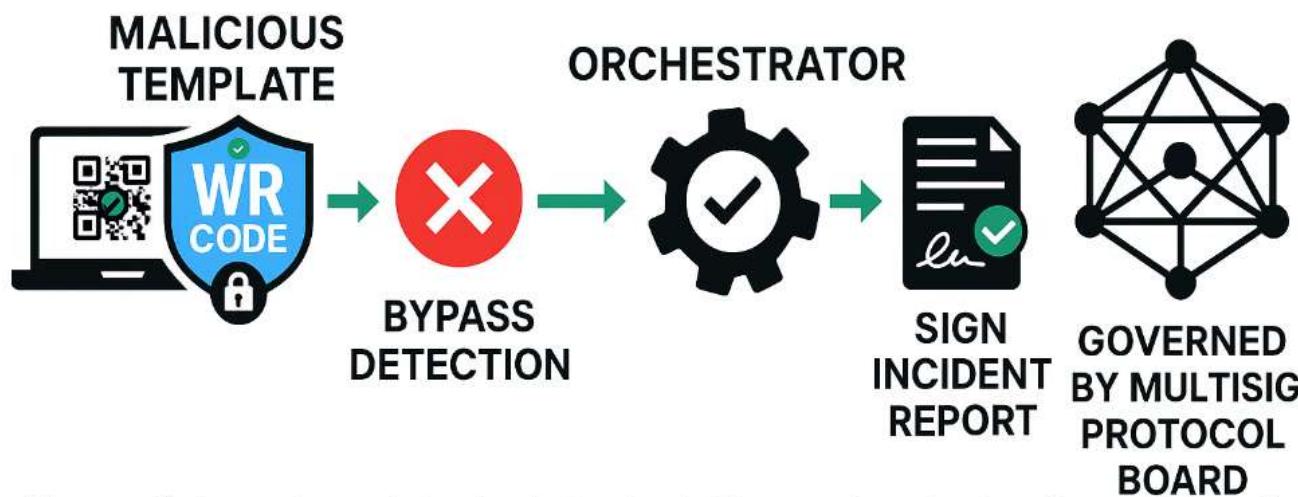
As robots grow more intelligent — in some domains exceeding human capacity — auditable decision trails, tamper-proof orchestration, and built-in kill-switches will become

inevitable safeguards. They are the mechanisms that ensure this acceleration does not spiral into disaster, but instead builds a robotic ecosystem that is verifiable, trustworthy, and aligned with human safety. As autonomous systems advance into mission-critical domains — from self-driving cars on public roads to humanoid robots in workplaces, hospitals, or even domestic robots in kitchens handling knives and other sharp tools — the absence of accountability mechanisms becomes increasingly unacceptable. Today, many of these systems still operate without fully auditable, transparent, and tamper-proof rules. Malfunctions are too often treated as unavoidable incidents, with logs reconstructed after the fact and oversight left to selective disclosures.

WRCode, together with its Proof of Execution mechanism, seeks to change this. It is being developed to anticipate the emerging era of autonomy and to pioneer the safeguards these systems demand. Every orchestration plan, decision, and execution step will follow WRCode guidelines and be anchored, auditable, and verifiable. Systems that are not registered, verified, and anchored will simply not run. If manipulation is attempted, or if a robot diverges from its validated template — whether it is an autonomous car on the road or a household robot in a kitchen with a knife — deterministic safeguards and built-in kill-switch logic will immediately stop execution before harm occurs.

In this way, malfunctions and manipulations become traceable and accountable events rather than opaque failures. Mission-critical automation will no longer be a black box, but a transparent and trustworthy infrastructure, where every decision trail is provable and immutable. As autonomous technologies move into daily life, tamper-proof orchestration is not optional — it must become the cornerstone of safety, compliance, and public trust.

BYPASS DETECTION



If a malicious template is detected, the orchestrator flags it locally and may submit a signed incident report. Revocation is governed through decentralized policy

Privacy-Preserving Ads in the Orchestrator

WR CODE introduces a new way to handle personalized advertising: specially trained, locally hosted LLMs decide which ads appear inside the agent boxes. This ad-selection pipeline runs entirely on-device. Even if a user opts to run other tasks via powerful cloud AI, the choice of personalized ads is always determined locally in a privacy-preserving way.

Because the system is transparent, open and built on a zero-trust architecture, anyone can verify how ads are processed. The orchestrator fetches anonymized ad packages (creative + metadata), evaluates them locally against on-device context, and never exposes user-level signals to external providers.

To make this process fully trustless, the WR Stamping system enforces runtime integrity: only software components that pass cryptographic verification are allowed to run. This enables not only for users but also for advertisers that the ad-delivery logic cannot be tampered with.

The result merges privacy, transparency, and sustainability: relevant ads can support free-tier users, and expensive API usage can be cross-financed through ads, reducing costs and making advanced AI services more affordable — all without compromising privacy or trust.

 **Strategic Trade-Offs and Professional Target Group**

Optimando AI's WR CODE architecture is designed around **autonomy, data privacy, and advanced user-controlled automation**. As a concept, it acknowledges several strategic trade-offs that professional users and potential investors should be aware of:

- **Higher Token and Context Costs:** Workflows involving large prompts, multi-agent logic, or external reasoning (e.g., cloud LLMs) can lead to increased token usage. However, these costs are steadily decreasing as open models improve.
- **Local Resource Requirements:** Running local LLMs and orchestration logic requires capable hardware (e.g., sufficient RAM and CPU/GPU). While often more efficient than cloud alternatives for burst workloads, continuous usage still implies measurable local energy use.
- **Advanced Setup Needs:** The system assumes a professional or technically literate user base. Users manage their own storage, security (e.g., encryption), and model selection.

These are not flaws, but deliberate design trade-offs in favor of **data control, workflow transparency, and security**. Most cloud-first systems offer convenience but at the cost of vendor lock-in and profiling risks. Optimando.ai is conceptually positioned for **entrepreneurs, researchers, educators, and technically skilled professionals** — users who gain long-term strategic value from maintaining control over their data and automation stack. For these users, the **advantages clearly outweigh the operational complexity**.

Evaluation and Distinction:

Novelty of optimando.ai's Approach

Given the landscape above, **optimando.ai's** combination of features **does appear to be novel and unmatched**. In particular, no known system offers *all* of the following in one package:

- **Fully local, browser-native multi-agent orchestration:** Many systems run in the cloud or require server components. Those that are local (browser extensions like Nanobrowser or RPA tools) don't typically orchestrate multiple autonomous agents across several browser tabs without user direction. Optimando's design of a local master tab coordinating slave tabs for different subtasks is unique.
- **By integrating directly into the browser landscape**—the primary interface for digital activity worldwide—optimando.ai enables real-time optimization or intelligent, context-aware, goal-driven optimization suggestions at scale, putting AI orchestration into the hands of every user without relying on proprietary platforms, cloud dependencies, or specialized infrastructure
- **Autonomous, proactive assistance:** Most current solutions are *reactive*. They await user queries or commands. A system that observes the user's context and proactively generates suggestions or carries out optimizations (e.g. automatically augments your task flow across multiple sites) is not mainstream yet. Yutori's "Scouts" come close conceptually (monitoring in the background)github.com, but those operate on specific user-defined goals (like watching a particular site or alert type) **rather than**

generally optimizing any ongoing browsing activity. An *AI that feels like a colleague actively helping unprompted* is largely aspirational right now.

- **Multi-agent parallelism in a user-facing application:** While research and some closed prototypes leverage multiple agents in tandem, typical user-facing AI assistants are single-agent. Optimando.ai's vision of parallel agents (each potentially with specialized roles or focusing on different tabs) coordinating in real time to help the **user is cutting-edge**. We see early signs of this in Opera Neon's ability to multitask and in Nanobrowser's planner/navigator duo, but these are either constrained or not autonomous. **No product fully utilizes a swarm of browser-based agents to continuously adapt to what the user is doing.**
- **Context-aware cross-tab optimization:** This implies the system maintains a high-level understanding of the user's objectives across multiple browser tabs or tasks. None of the surveyed tools truly does this. For instance, if a user is doing research with several tabs, current AI assistants might summarize one tab at a time when asked, but they won't *on their own* consolidate information from all tabs or reorganize them for the user's benefit. **Optimando.ai** aiming to provide "**real-time, context-aware optimization**" suggests it would do exactly that – **something quite novel**.

In conclusion, the core architecture of **WR CODE** – a local master–slave tab framework enabling an autonomous multi-agent assistant system – **is novel**. Existing systems offer pieces of the puzzle (cloud-based autonomy, local extensions, multi-tab tools, etc.), but none delivers the same integrated experience. Therefore, **optimando.ai's** implementation would represent a distinct advancement in browser AI orchestration and autonomy. Its closest peers (Google's Mariner, OpenAI's Operator, Opera's Neon, academic

Orca, and various extensions) each lack at least one crucial element (be it full local execution, source-available, proactivity, multi-agent parallelism, or deep context integration). As such, optimando.ai's concept stands out as unmatched in combining all these features into one system, **marking a potentially significant innovation in the AI browser assistant space.**

- **Blockchain Certification of AI Factsheets (IBM Patent, 2022):** IBM researchers patented a method to certify AI model factsheets (documentation of model details and performance metrics) using blockchain. In this system, an AI model's factsheet is generated and then anchored on a blockchain, producing an attestation certificatepatents.google.com/patent/US20220033001A1. The blockchain link serves as a tamper-evident record certifying the model's training data, testing results, and validation metrics. The patent describes using a smart contract as a moderator and certifying authority for AI model marketplaces, ensuring that any changes to a model's factsheet (e.g. updates to metrics) are tracked immutablypatents.google.com/patent/US20220033001A1. This approach provides a decentralized trust layer for AI model governance, preventing factsheet tampering and enabling reliable verification of an AI's performance claims.
- **Decentralized AI Model Deployment with NFTs (US Patent 11,494,171, 2022):** This patented platform uses blockchain tokens to orchestrate AI model publishing, validation, and deployment. Each AI model is represented as a non-fungible token (NFT) on a blockchain, embedding the model's hash and a URI pointing to its storage locationpatents.google.com/patent/US11494171B1. Publishing a model as an NFT creates an immutable record of the model version, and validators (authorized QA engineers or automated agents) vote on the model's performance by testing it against benchmarkspatents.google.com/patent/US11494171B1. A genesis block is created for the model's NFT

containing its hash and metadata; once a quorum of validators reach consensus on its accuracy, a new block is added to confirm the model's validitypatents.google.com. This system combines AI orchestration with tamper-proof blockchain records: it ensures that models cannot be swapped or altered unnoticed, and it leverages decentralized consensus to approve model quality. Notably, it also supports distributed deployment of models on a network of computing providers, recording each deployment and update on the ledgerpatents.google.com.

- Blockchain Trust Systems for Predictive Analytics (Strategemist Patent, 2025): A recent patent introduces a decentralized predictive analytics framework integrating blockchain, federated machine learning, and advanced cryptographystrategemist.com. It uses distributed ledger technology (DLT) to achieve tamper-resistant AI workflows: model updates and training contributions from multiple parties are logged on a blockchain, and the system leverages zero-knowledge proofs (ZKPs) and homomorphic encryption to validate model computations without revealing private datastrategemist.com. The framework includes quantum-resistant cryptography and Byzantine fault-tolerant consensus to ensure that model parameters and training data remain secure and *verifiable* across untrusted participants. In practice, this means each federated learning update can be accompanied by a cryptographic proof of correctness (e.g. a ZKP) and written to an immutable ledger. The patent highlights features like immutable model audit logs on-chain, *smart contracts* enforcing governance policies for model updates, and a “federated AI trust score” to rate contributorsstrategemist.com. By anchoring AI model lineage and updates on a

ledger, the system enables trustless verification of AI decisions and model integrity, addressing issues of data tampering, adversarial poisoning, and compliance in multi-party AI deployments.

- Academic Framework – Logging AI Decisions to Blockchain (Kulothungan et al., 2025): In the paper *“Using Blockchain Ledgers to Record AI Decisions in IoT”*, researchers propose a blockchain-based audit trail for AI-driven decisions in IoT systems mdpi.com. Every AI inference made at the edge (e.g. an anomaly detected by a sensor or an autonomous control action) is cryptographically signed by the device and recorded as a transaction on a permissioned blockchain. Each log entry includes the input data, the AI model ID or version, and the output decision, hashed or encrypted as needed for privacy mdpi.com. This creates an immutable, timestamped ledger of the AI’s decisions, which stakeholders or regulators can later audit. The system guarantees non-repudiation (devices cannot deny the decisions they made) and integrity (any attempt to alter a past decision would be evident on the chain). The authors demonstrate use cases in healthcare (logging diagnostic AI alerts) and industrial control, aligning with emerging AI governance regulations that require traceability (e.g. the EU AI Act’s logging mandate) mdpi.com. By anchoring decision provenance on blockchain, this approach increases transparency of AI workflows and allows independent verification that a given AI decision followed the intended model and data inputs.
- Prove AI Platform (2023–2025, Hedera DLT): *Prove AI* is a commercial AI governance platform that uses a distributed ledger (Hedera Hashgraph) to provide tamper-proof oversight of AI models in production. It creates an auditable trail of all key AI lifecycle events – from training data used, model versions deployed, to inference outputs – and stores hashes of these events on the ledger proveai.com. By doing so, *Prove AI* ensures that any change in a model or any decision made

by the AI is recorded in an immutable log that auditors can trust. The platform is designed to help organizations comply with AI regulations and “break open the black box,” in partnership with IBM’s AI governance tools proveai.com. It anchors AI metadata (like datasets, model parameters, and decisions) on Hedera, and uses digital signatures to guarantee authenticity of each event. This ledger-based approach enables verification of AI workflows post-hoc: for example, a company can cryptographically prove which dataset version was used to train a model, or demonstrate that an AI’s output in a given case wasn’t altered. By hashing and timestamping events, and leveraging Hedera’s consensus, Prove AI delivers a tamper-evident record of AI operations, bringing trust and accountability to complex AI pipelines.

- Zero-Knowledge Proofs for Verifiable ML (ZKML Research): A growing body of work applies zero-knowledge proof techniques to machine learning so that a model’s execution can be verified without revealing its inputs or parameters. In essence, ZKML allows one party to prove that “*a certain ML computation was performed correctly*” to another party, without that verifier needing to run the model themselves. For example, recent frameworks allow a prover to demonstrate that they correctly executed a neural network inference on given data, or that a model was trained to achieve a certain accuracy, all by generating succinct cryptographic proofs. A 2025 survey by Peng *et al.* reviews these developments, noting that ZKP technology can validate model performance and authenticity in training and inference without disclosing sensitive data arxiv.org. Solutions like zk-SNARKs have been used to prove the correctness of predictions, ensuring an AI service can be trusted even if run on external infrastructure. One prominent example is *zkML* for neural networks, where the model’s computations (matrix multiplications, activation functions) are translated into

arithmetic circuits that can be verified on-chain. This guarantees integrity of AI decisions: a model provider can't lie about an output, because the proof would fail. However, a major challenge has been efficiency – e.g., proving a large deep network's inference can incur *1000x overhead* in time and memory [arxiv.orgarxiv.org](#). Efforts like *zkCNN*, *zkSNARK-optimized networks*, and *zkLLM (Zero-knowledge for LLM inference)* [arxiv.org](#) are pushing the frontier, enabling privacy-preserving yet verifiable AI, where even the model weights can remain hidden but the result is assured. In summary, ZKML techniques provide cryptographic guarantees of an AI pipeline's correctness, anchoring trust in math rather than in centralized auditors – but they currently work best for smaller models or portions of a pipeline due to computational costs.

- Optimistic Verification for AI Pipelines (opML and Related Work): Inspired by optimistic rollups in blockchain, *optimistic verification* schemes assume that an AI computation is correct by default and only perform an expensive check if someone disputes the result. One notable implementation is opML (Optimistic Machine Learning on Blockchain, 2024), which uses *fraud proofs* to verify ML results [arxiv.org](#). In opML, an AI model inference is executed off-chain (for performance) and the result is posted on-chain. The system then enters a challenge window during which any validator can question the result. If a challenge arises, opML performs an interactive bisection protocol (similar to Arbitrum's approach for smart contracts) to pinpoint the exact step of the ML computation that differs from the claimed result [arxiv.orgarxiv.org](#). Essentially, the large computation (e.g., a sequence of tensor operations) is split into smaller segments to find where the miscompute occurred, and a minimal critical step is verified on-chain to prove the result was wrong. If the prover is found cheating, they lose a stake (penalizing bad actors), otherwise the result is accepted after the

challenge period. Conway *et al.* report that opML can run a 13-billion-parameter model's inference in a decentralized network with low cost by avoiding upfront ZK proofs [ethresear.charxiv.org](#). Another system, Agatha (Zheng et al., 2021), was an earlier optimistic scheme focusing on DNN inference verification with negligible on-chain overhead: it too used an interactive on-chain game to catch incorrect results with minimal performance [hitarxiv.org](#). The optimistic approach provides "AnyTrust" security [arxiv.org](#) – as long as at least one honest validator is watching and capable of recomputing the AI task, any fraud will eventually be exposed. The advantage over ZK proofs is efficiency (no huge proof to generate if all act honestly), but the trade-off is latency (one must wait through the dispute window, which could be minutes) and reliance on game-theoretic incentives for watchers [medium.commedium.com](#). This method is well-suited for complex AI workflows where full ZK proof is infeasible; it optimistically assumes correct execution and only occasionally requires on-chain arbitration, thereby anchoring trust via economic incentives and interactive verification rather than heavy cryptography.

- Consensus and Voting-Based Verification (spML and Cryptoeconomic AI): A third approach to verifiable AI workflows is distributed consensus or voting among multiple AI agents. In 2024, Zhang *et al.* proposed *spML*, a framework where a decentralized network of validators all run the same inference and vote on the correct result [arxiv.org](#). If a majority (or supermajority) agree on the output, it is accepted; if there's a disagreement, it signals potential tampering or error. This is akin to a *cryptoeconomic consensus*: the validators might stake tokens and be rewarded or slashed based on whether they align with the majority outcome (deterring them from random or malicious results). Such a scheme was described conceptually as "*cryptoeconomic ML*" [medium.commedium.com](#), where

the user can request N independent nodes to perform an ML task – if their results differ, a simple voting (or stake-weighted voting) decides the outcome and penalizes outliers. The benefit of this approach is that it's fast (low latency) – essentially running in parallel and just requiring a commit-reveal or aggregation step on chain[medium.com](#). It doesn't require heavy cryptography or complex games; however, its trust model is weaker. It assumes a honest majority of validators, so a collusion of enough nodes could still deceive (similar to oracles). Projects like Ora and Gensyn in 2024 were exploring this space by creating marketplaces of AI compute where multiple providers execute tasks and cross-verify results[medium.com](#). While not as mathematically guaranteed as ZK or optimistic proofs, consensus-based verification provides a practical layer of defense: it makes cheating economically difficult (you'd need to corrupt many nodes) and offers a sliding scale of assurance (more validators can increase confidence). This approach can be combined with ledger anchoring – e.g., each node's result hash is recorded on a blockchain, and a smart contract tallies votes and handles rewards/slashing. In sum, voting-based orchestration introduces redundancy and game theory to verify AI workflows, trading absolute certainty for efficiency and scalability in real-world deployments.

- **IOTA Tangle for Data Integrity in Automation:** Beyond blockchains, Directed Acyclic Graph (DAG) ledgers like IOTA's Tangle have been applied to verify workflow integrity, especially in IoT and data automation scenarios. IOTA's DAG architecture offers fee-less transactions and high throughput, which makes it suitable for recording numerous sensor readings or device actions for audit. For example, IOTA introduced Masked Authenticated Messaging (MAM) channels to allow IoT devices to stream data with integrity proofs. In such a setup, each device publishes hashes or digital signatures

of its data packets to the Tangle, and because the Tangle is append-only and secured by cryptography, any consumer of the data can verify it hasn't been tampered with. One illustrative use-case (Feng, 2018) showed how a sensor can log each reading's hash on IOTA, enabling any downstream system to check that the reading was not altered by comparing it to the Tangle entry <feng.lufeng.lu>. The advantages of using IOTA/DAG for workflow verification include zero transaction fees, scalability, and offline tolerance. With no miners to pay, even tiny devices can afford to anchor each event (e.g. 1000 transactions/minute) on the ledger <feng.lufeng.lu>. DAG consensus (in IOTA's case, a coordinator or weight-based approval) confirms the order and integrity of events without the energy cost of proof-of-work. This has been leveraged in real-world automation, for instance supply chain tracking where each handoff is logged via a QR code scanned and recorded on IOTA, or smart energy grids where device commands are signed and traced on the Tangle. While DAG-based systems are somewhat newer and require careful security analysis, they represent a promising avenue for lightweight, scalable verification of AI-driven workflows – essentially providing the benefits of blockchain (immutability, transparency) without the bottlenecks, thus suitable for high-frequency or micro IoT events that AI systems often produce <feng.lufeng.lu>.

QR Code–Triggered AI Execution Flows

- QR Codes as Triggers in Automated Workflows (Supply Chain Example): QR codes are increasingly used as physical-digital bridges to initiate automated processes. A prominent example is Morpheus.Network's supply chain platform (launched 2019), which uses QR code scans to trigger workflow steps on a blockchain-backed system <news.morpheus.network>. In this platform, logistics

documents or shipping containers are tagged with QR codes; when a user (or IoT scanner) scans the code at a checkpoint, it automatically triggers a predefined workflow in the system – for instance, notifying stakeholders, updating an item's status, or releasing a payment. These events are recorded as “Digital Footprints” on a distributed ledger for transparencynews.morpheus.network. The QR code essentially serves as a real-world trigger to a smart contract or AI agent: for example, scanning a code at a warehouse could invoke an AI vision system to inspect goods, or prompt an RPA bot to update inventory. Because each scan is logged immutably, the system achieves tamper-proof workflow execution – it’s evident if a step was done out of order or not at all. This combination of QR codes + blockchain is particularly powerful in multi-party processes like supply chains, where no single entity is fully trusted. The QR codes provide a user-friendly way to kick off digital actions (no need to navigate apps or interfaces in the field), and the backend ledger ensures the resulting AI or automated actions are verifiable and traceable. Overlaps with AI come when these triggers invoke AI services – e.g., automatic customs document checking, or routing optimization algorithms – whose decisions can be anchored to the workflow log. The Morpheus.Network case demonstrates how QR codes can integrate with AI orchestration and DLT to facilitate real-world automation that is both convenient and auditable.

- “AI-Enabled QR Codes” for Consumer Engagement (Harsha Angeri, 2023): In a tech demonstration, Harsha Angeri showcased how scanning a simple QR code can *instantiate a complex AI agent workflow in the cloud*medium.datadriveninvestor.commedium.datadriveninvestor.com. The concept is that any physical object (a product, poster, appliance, etc.) can have an “AI agent” counterpart that awakens when its QR code is scanned. For example, scanning a T-shirt’s QR code could trigger an AI

agent that gathers product info, compares prices and styles, and then opens a chat with the user to answer questions about that T-shirt medium.datadriveninvestor.com/medium.datadriveninvestor.com. Technically, the QR code scan sends a request to a cloud function or webhook which launches a series of AI tasks – such as calling a language model, accessing databases or web APIs, and orchestrating a response back to the user (in Angeri's demo, via a Discord chat message). The QR code triggers a cloud function that runs the AI agent, pulling live data from the internet and even chaining through automation services (like Zapier) to deliver results to the user medium.datadriveninvestor.com. This effectively makes a static object “intelligent” on demand. It also enables workflow composition via QR: one QR scan can invoke multiple AI services. In one example, scanning a QR led to three different AI agents being called in sequence – one doing image recognition, another (BabyAGI) planning a task list, and a third generating new images for design ideas medium.datadriveninvestor.com/medium.datadriveninvestor.com. Such frameworks combine AI orchestration, RPA, and human-in-the-loop in a lightweight manner: the human simply scans a code and converses, while behind the scenes a managed workflow coordinates AI APIs and possibly human fallback (if needed). While this is a prototype-level illustration, it highlights a broader trend: QR codes are being used as physical triggers for digital AI workflows, from customer service bots that start when you scan a product's code, to industrial maintenance AIs that launch diagnostics when you scan a machine's code. The simplicity of QR scanning lowers the barrier for users to invoke complex services, and when tied to robust backend orchestration (potentially with ledger verification of each step), it creates a seamless yet trustworthy automation. Companies are beginning to integrate similar ideas – for instance, some customer support systems now have QR codes on devices that, when scanned, start an automated troubleshooting chat with an AI, and IFTTT in 2025

introduced QR scan triggers so users can easily link a code to any automated applet on their phone ifttt.com. This fusion of physical QR triggers with AI and blockchain opens up novel use cases (sometimes dubbed “phygital” experiences), ensuring not only convenient activation of services but also that those services can be verified and audited in retrospect when tied into cryptographic workflow logs.

- **NFTs and Triggers in Workflow Automation:** Non-fungible tokens are mostly known in digital art and collectibles, but they have also been experimented with as triggers or access tokens for real-world and AI-driven workflows. For example, some proposals use NFTs as digital keys that, when scanned (via a QR or an RFID associated with the NFT), trigger an action or grant access to a service. One patent from 2021 outlines using an NFT-based token to manage digital rights in a decentralized content network, where accessing the content triggers a smart contract check of the NFT ownership patents.google.com. In an AI workflow context, one could imagine an NFT representing a permission or subscription that, when presented (scanned as a QR code linking to the token), launches an AI service if the token is valid. This ties into human-in-the-loop and tamper-proof workflows: an NFT could encapsulate a user’s consent or a human approval step, which the AI pipeline must verify cryptographically before proceeding. While specific patents combining *all* these elements (AI orchestration + NFT + human/RPA) are still emerging, we see building blocks in public projects. For instance, Chainlink’s dynamic NFTs allow off-chain events (like achieving a certain AI model outcome) to update an NFT’s state chain.link, which in turn could trigger follow-on processes. In RPA (Robotic Process Automation), enterprise workflows could use an NFT-like token for each task instance – say a token moves through a workflow, getting signed by human approvers and AI results

at each step, ensuring an immutable audit trail of a human-in-the-loop process. The convergence of these ideas is nascent but promising: by combining tokenization, cryptographic verification, and AI, one can create *self-governing workflows*. For example, a complex process (like a loan approval that involves an AI credit model and a human manager) might issue a token that the AI writes its decision to (signed and logged), then passes to the manager who adds a signature, and finally a smart contract automatically executes the loan disbursement when all signatures (AI + human) are present. Such multi-faceted systems are on the horizon – integrating NFTs for state tracking, AI for decision-making, and ledger tech for trust – to achieve tamper-proof, transparent AI orchestration with human governance. (While concrete unified implementations are still evolving, the individual components are being patented and developed as seen above.)

- DAG-based ledgers (IOTA/MAM) for AI/IoT verification: Some works exploit directed-acyclic-graph ledgers for data integrity in sensor/IoT pipelines. For example, IOTA's Masked Authenticated Messaging (MAM) provides an encrypted, sequential data stream: each message is chained by a root and confirmed by IOTA's consensus[medium.com](#). In practice, sensor readings (which could include ML or AI outputs) are sent through MAM channels; the IOTA Tangle's DAG ensures immutability and timestamps on these values. This has been proposed for critical infrastructure (e.g. dike monitoring): the IOTA consensus “ensures integrity within the data flow” so that any tampering with sensor or AI output can be detected later[medium.com](#). Similar DAG ledgers could be used to timestamp and verify workflow events or ML results in a lightweight, fee-free manner.

Sources:

1. US Patent 11,483,154: *Artificial intelligence certification of factsheets using blockchain* (IBM, granted 2022)patents.google.com/patents/US11483154.
2. US Patent 11,494,171: *Decentralized platform for deploying AI models* (2022) – using NFTs and blockchain for AI model validationpatents.google.com/patents/US11494171.
3. Strategemist Corp. – *Blockchain Trust Systems* patent overview (2025)strategemist.comstrategemist.com.
4. Kulothungan, V. (2025). “Using Blockchain Ledgers to Record AI Decisions in IoT.” *IoT* 6(3):37mdpi.commdpi.com/2696-940X-6-3-37.
5. Prove AI – Product description (2023) – Tamper-proof AI governance on Hederaproveai.com.
6. Peng, Z. et al. (2025). “A Survey of Zero-Knowledge Proof Based Verifiable Machine Learning” (arXiv 2502.18535)arxiv.org/abs/2502.18535.
7. Conway, K. et al. (2024). “opML: Optimistic Machine Learning on Blockchain” (arXiv 2401.17555)arxiv.org/abs/2401.17555; Zheng, Z. et al. (2021). “Agatha: Smart Contract for DNN Computation” (arXiv 2105.04919).
8. Qureshi, H. (2024). "Don't Trust, Verify: An Overview of Decentralized Inference" (Dragonfly Research)[medium.commedium.com](https://medium.com/commedium/don-t-trust-verify-an-overview-of-decentralized-inference-dragonfly-research-103a2a2f3a).

9. Morpheus.Network (2019). *Supply Chain Platform Launch* – QR code triggers and blockchain footprints in workflows news.morpheus.network.
10. Angeri, H. (2023). "Every Object is Intelligent – Enabling All Objects with AI Agents" (Medium) [medium.datadriveninvestor.com](https://medium.datadriveninvestor.com/medium.datadriveninvestor.com)
11. Feng, X. (2018). "Data Integrity and Data Lineage by using IOTA" – blog demonstrating IOTA's use in IoT data verification feng.lu.

Google DeepMind's Project Mariner techcrunch.comtechcrunch.com, OpenAI's Operator techcrunch.comtechcrunch.com, UCSD's Orca browser research arxiv.orgarxiv.org, the Nanobrowser open source project github.comgithub.com, Opera's AI initiatives (Aria and Neon) press.opera.compress.opera.com, and curated lists of web AI agents and automation tools github.comgithub.com. Each of these informs the feature comparison and underscores the novelty of optimando.ai's approach.

 Novel Innovations of *optimando.ai*

 Real-Time Contextual Optimization Layer

A source-available, browser-based orchestration tool that continuously interprets user behavior, screen context, and intent to dynamically route tasks to specialized AI agents — enabling real-time assistance without requiring explicit commands.

optimando.ai proactively supports the user based on their current activity — such as drafting, summarizing, or researching — rather than relying solely on typed input.

Users can initiate multi-step automations via natural language, speech commands, or automatically detected user intent — including instructions to display results in designated visual slots within the interface. Display slots can also be allocated dynamically based on contextual relevance and priority, allowing high-urgency or decision-critical outputs to automatically surface in more prominent or persistent positions.

The system responds dynamically by monitoring the workspace (e.g. tab content, interaction patterns) and coordinating agents in real time.

This represents a **novel source-available approach to AI-driven workflow orchestration**, combining live context awareness, voice-driven control, and visual task management in a unified, browser-centered environment.

.🛡️ Integrated Privacy Layer with Risk Mitigation

On-device logic detects sensitive content, delays or masks prompt execution, and can optionally inject decoys to obscure real user intent.

This layered approach to privacy — combining detection, execution control, and obfuscation — is not present in mainstream tools. It addresses profiling risks directly at the orchestration level, which most cloud- or agent-based systems do not offer.

Feature Comparison

| Feature | Proposed Framework | Opera Neon | Google Project Mariner |
|---|--|------------|------------------------|
| Realtime backend optimization suggester | Yes (context-aware AI suggestions tied to global user goals) | No | No |
| Multi-agent coordination | Yes (multiple slaves per tab) | Partial | Partial |
| Browser tab as agent | Yes (users tag tabs) | No | No |
| Source-available | Yes (user-run) | No | No |
| User-controlled LLM selection | Yes (any open or cloud LLM) | No | No |
| Data sovereignty | High (all local, opt-in cloud) | Medium | Low |
| Man-in-loop by design | Yes | Yes | Yes |
| Unique agent IDs (multi-user) | Yes | No | No |
| Multitasking / parallel tasks | Yes (multiple slaves) | Yes | Yes |

Air-Gapped, Multi-Agent Orchestration (No Cloud Dependency)

A fully local deployment option where multiple AI agents run in coordination without internet access, suitable for high-security or regulated environments.

While platforms like **LangChain**, **AutoGPT**, **Cognosys**, **Lindy.ai**, **Superagent**, **AgentOps**, **Microsoft AutoGen**, and various open source agent frameworks offer components such as agent coordination, multi-step automation, or tool integration, they typically lack one or more of the truly distinctive features that *optimando.ai* provides.

In particular, while some tools offer local model integration or agent chaining, *optimando.ai* uniquely provides a **source-available, plug-and-play multi-agent orchestration system** that can run **fully offline if needed** — including **real-time context interpretation**, **contextual intent support**, **speech-triggered automation**, and a **visual slot-based interface** for task execution.

This combination of capabilities is **currently unmet** in other AI automation offerings, positioning *optimando.ai* as a **novel solution for privacy-sensitive, highly interactive, and locally controlled AI-driven workflows**.

These three features — especially in **combination** — position *optimando.ai* as a **privacy-focused, locally-executable orchestration system with intelligent agent coordination**, which is **currently unmatched** in the browser-based LLM tooling space.

Unlike Mariner or Orca, optimando.ai does not wait for the user to act. It acts with the user, continuously enhancing workflows as they unfold—across tools, content types, and digital services. It is not a helper or assistant. It is a real-time orchestration layer for the modern web workspace.

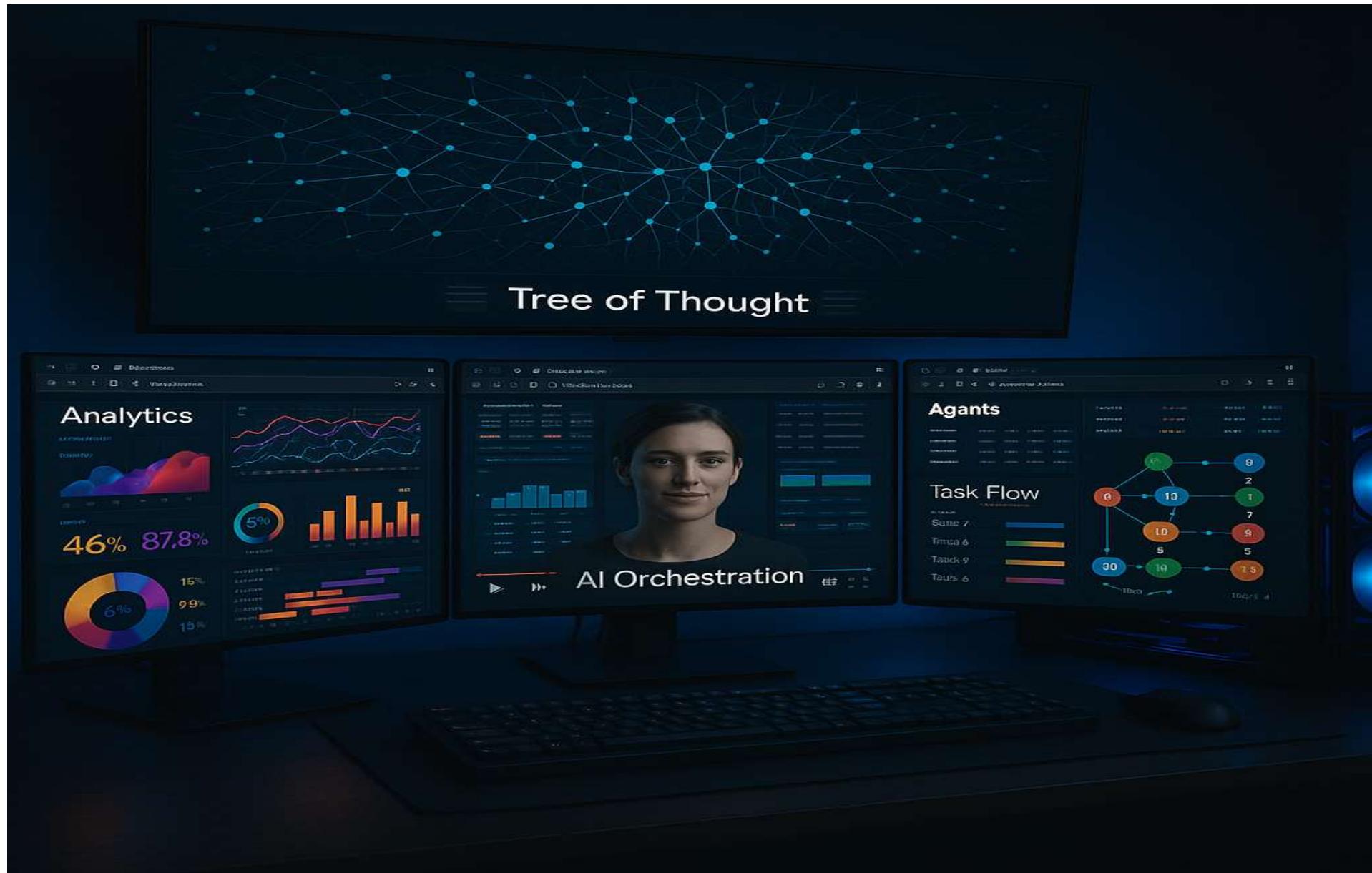
To our knowledge, no existing system—academic or commercial—combines autonomous multi-agent orchestration, tab-based modularity, proactive real-time augmentation, and privacy-first, local execution. This positions optimando.ai as a breakthrough platform in browser-native AI automation.

🎮 From Real-Time Gaming to Real-Time Intelligence: A Paradigm Shift

Modern gaming has become a proving ground for real-time computing performance. Today's top-tier systems deliver:

- 🎮 **144–360 FPS forward rendering,**
- ⏱️ **Sub-10ms input-to-photon latency,**
- 🤖 **DLSS/Frame Generation by AI,**
- **and instant asset streaming over hybrid cloud-edge setups.**





High-Fidelity, Real-time Optimization Towards Predicted or Defined Goals

These same principles—low-latency responsiveness, forward-thinking, dynamic rendering, and distributed compute—are now crossing into productivity and automation. Breakthroughs on multiple levels will make unimaginable things possible within the next decade and this conceptual browser orchestration framework puts this power into the hands of everybody with a pc and internet access. After all advanced AI-driven fast-pace gaming compute is similar to real-time data generation.

Imagine a knowledge worker's future desktop setup:

-  **Screen 1:** A browser helper tab hooked to an LLM refines every question and interaction you write—augmenting your thinking through prompt optimization and chain-of-thought amplification.
-  **Screen 2:** Another tab visualizes live data from an internal MCP (Multi-Channel Processing) server, rendering interactive, high-frequency charts in real time using forward-rendering browser tech via WebGL or WebGPU.
-  **Screen 3:** A helper agent watches your actions and assembles a narrated video tutorial using generative AI—documenting decisions, insights, and process flows as you work.

Responsive Output via Smart Buffering

To ensure a fluid user experience, the output coordinator buffers AI results before displaying them in visual slots. Only complete, relevant outputs are shown, keeping the interface responsive and distraction-free. As

AI performance and computation capabilities improves over time, this buffer time will shrink — moving the experience closer to true real-time interaction, even in complex multi-agent scenarios.

Unlike siloed platforms, WR CODE empowers users to compose, tailor, and share interactive multi-agent workflows—where different AI tools not only coexist, but actively build on each other's output to deliver the best possible result, without any vendor lock-in.

❖ The Technical Foundation

Unlike traditional agent systems that rely on centralized cloud logic or bespoke SDKs, the optimando.ai framework is built on:

- Browser-native orchestration using tabs, not containers
- DOM-level prompt injection and readback, per desktop/mobile app and browser extension
- User-defined session templates and context-aware routing logic
- Customizable update intervals (DOM completion, screenshot loop, stream window)
- Autonomous or manual feedback triggers, including peer-to-peer helper tab interactions
- Full MCP server compatibility via orchestrator logic via helper tab (local/remote event listeners)
- Hybrid LLM handling, where each helper tab can run:
 - Local models (e.g., Mistral, LLaMA, Phi-3)

- Cloud models (e.g., GPT-4, Claude, Gemini, Deepseek, Mistral)
- Or even autonomous agents (e.g., OpenDevin, Project Mariner)
- Security by Design through browser sandboxing, session isolation, and non-invasive architecture
- The browser, long seen as a passive interface, is now the orchestrated runtime layer.
- Security by Design: The system leverages native browser sandboxing, session isolation, and a non-invasive architecture (no root access, no background daemons), reducing attack surfaces and simplifying compliance.
- Modern autonomous agents—such as OpenDevin, Google’s Project Mariner, or Baidu’s Ernie Bot Agent—highlight the global trend toward AI-driven process delegation. However, many existing solutions remain closed, platform-bound, or require deep system integration. Optimando.ai takes a more flexible route: its browser-based helper tab concept allows users to integrate AI agents and automation tools—including LLMs, n8n, Zapier, Make, or other cloud/local services—without leaving the familiar browser environment.
- This architecture enables real-time orchestration of AI workflows and brainstorming sessions across browser tabs, supporting both cloud-based APIs and fully local execution. It offers a modular and scalable framework that allows organizations to incrementally adopt AI-driven automation—without lock-in, without compromising data ownership, and with minimal infrastructure requirements. The result is a powerful, interoperable AI workspace that aligns with existing digital behavior while opening the door to highly personalized and responsive task automation.

The Browser as a Universal AI Gateway

Why the browser? It is universally available, cross-platform, and runs on every device

- **It has evolved with WebGPU, Service Workers, Security Sandboxing, and full user-level isolation**
- **It allows interaction with any digital tool or AI system that exposes a UI**
- **It is where 98%+ of digital work happens—from cloud IDEs to enterprise dashboards**

optimando.ai leverages this to orchestrate entire multi-agent workflows from within the browser, controlled by a single orchestrator app on your device and a browser addon. No need for server-side logic—only tabs. For users seeking maximum privacy and control, the orchestration tool can be installed on a bootable, encrypted SSD preconfigured with a hardened Linux distribution such as Ubuntu. This setup allows the entire orchestration environment—including the browser-based agent system and supporting components—to run in an isolated, portable, and tamper-resistant workspace.

To simplify deployment, Optimando.ai will offer a ready-to-use secure setup, which includes full disk encryption, pre-installed orchestration software, and optional integration of local language models (LLMs) for fully offline workflows. This approach is ideal for professionals, researchers, or organizations that require both AI automation and strict data sovereignty.

Secure Deployment via Bootable Encrypted SSDs

To support privacy-focused and offline use cases, the orchestration framework can be deployed on a bootable SSD with full-disk encryption, running a desktop-capable Linux distribution such as Ubuntu 22.04 LTS Desktop Edition. This configuration allows the entire orchestration system—including the browser-based interface, coordination logic, and helper agents—to run in a graphical, self-contained, and tamper-resistant environment. The inclusion of a full desktop environment is essential, as it provides the graphical interface needed for interacting with browser tabs, multi-agent outputs, and visual workflows—similar to how a typical end-user system operates (e.g., on Windows or macOS).

For maximum security, the entire device should be encrypted using technologies like LUKS full-disk encryption, ensuring that no part of the disk remains exposed to boot-time malware or unauthorized data access. The system image can also be cryptographically hashed to enable post-installation integrity checks and reproducible builds. Additional hardening measures such as secure boot and optional air-gapped operation may be applied depending on the threat model.

This deployment strategy is particularly useful in environments with strict data governance policies, limited or no internet access, or where offline, local AI processing is required.

Intellectual Property Disclaimer and Defensive Disclosure

This document is part of the broader **WR CODE architecture**, which includes but is not limited to multiple foundational innovations and inventions such as:

- **WRCode** (decentralized QR-based orchestration triggers)
- **WRVault** (secure, user-controlled execution vault)
- **WRPay** (trustless biometric payments enhanced by orchestrated AI workflows)
- **WRConnect** (secure context/session relaying and service bridging)
- **WRCollect** (loyalty, reward, and context-aware capture for automation)
- **WRStamped** (cryptographic timestamping and integrity verification)
- **WRGuard** (cryptographic runtime integrity check for WordPress and other software)
- **WRPass** (identity-linked access control for agent and service authorization)
- **QRGiraffe** (visual AI trigger and action binding tool for physical environments)
- **SecureEmail** (tamper-proof, agent-verified email composition and delivery system with integrated Solana verification)

These mechanisms together form a novel, privacy-first automation framework designed to function across both digital and real-world interfaces.

This publication is released as a **defensive disclosure** to:

- Establish **prior art** and prevent monopolization of the described methods

- Ensure that the architecture remains **open, auditable, and free to implement**
- Protect the original author from future claims of IP infringement

This work has been:

- **Cryptographically timestamped on Bitcoin** (via OpenTimestamps)
- **Published on Zenodo and GitHub**

No component of this system is being submitted for patent protection. Any future attempt to patent materially identical systems or core mechanisms without **clear technical deviation** may be challenged based on this disclosure.

WR Codes are open to everyone: any user can register software, products, machines, things, documents, media, or other assets and enrich them with context, memory, and multi-AI agent workflows. Private users can set up simple use cases—such as linking to trusted websites like YouTube or Amazon for downstream AI workflow automation—with restrictions.

Companies, however, always require a special company publisher account registration. For advanced scenarios—such as multimodal automation, opening multiple non-whitelisted website pages in a grid, app integrations, or orchestrating machine workflows—this verified company account is mandatory. It includes domain and DNS validation, ensuring that complex workflows remain secure and authentic while the system stays flexible and accessible for all users.

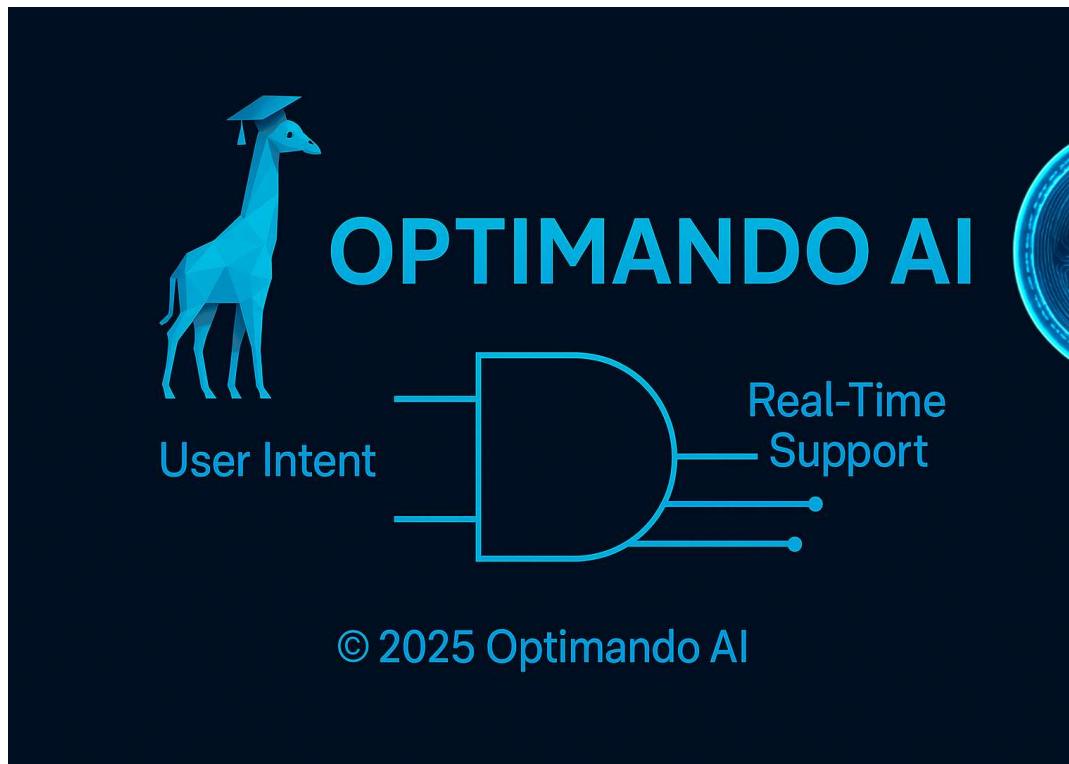
FLAG A Timestamped Innovation. First Public Release of Its Kind.

The conceptual design and technical outline of this system were first made publicly accessible in 2025 and cryptographically timestamped using [OpenTimestamps.org](https://opentimestamps.org) on the Bitcoin blockchain. This verifiable proof-of-publication ensures authorship precedence and protects against future claims of originality.

☞ To the best of our knowledge, this was the first publicly released source-available orchestration framework enabling browser-central, tab-based AI agents to coordinate, optimize, and suggest real-time strategies—across devices, sessions, and users. Unlike traditional automation tools that react only to explicit user input, this system is built around proactive, forward-thinking, continuous monitoring and intelligent feedback loops. Helper tabs autonomously observe context and suggest optimizations without requiring manual triggers—delivering a dynamic, adaptive AI experience across the user's digital workspace.

Generative AI was used solely for formatting, editing, and illustrative image generation; the underlying concept and contributions are original.

Concept Timestamped on Bitcoin



© 2025 Oscar Schreyer / O.S. CyberEarth UG (haftungsbeschränkt) — published as part of the Optimando.ai project.

This work documents the WR CODE© architecture, including QRGiraffe© — an source-available framework for browser-based AI orchestration developed by Optimando.ai.

The textual and conceptual content is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). Attribution and citation play a critical role in supporting open innovation by promoting transparency, crediting contributors, and enabling iterative, collaborative progress.

Certain components — such as source code, automation templates, or diagrams — may fall under different licensing terms depending on their category.
Please refer to the included license files for precise details.

- Software implementation (orchestrator, Augmented Overlay, WR Code integration, WRVault, WRGuard):
Licensed under the OpenGiraffe Protected License (OGPL-1.0) — a source-available license restricting redistribution, forking, SaaS deployment, and removal of system branding.

- Conceptual documentation (architectural descriptions, diagrams, workflow logic, explanatory text):

Licensed under Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0).

Proof of authorship for all included materials has been cryptographically timestamped using OpenTimestamps.

License references:

- Creative Commons (CC BY-SA 4.0): <https://creativecommons.org/licenses/by-sa/4.0/>
- GPL-1.0 (OpenGiraffe Protected License): included in this repository

