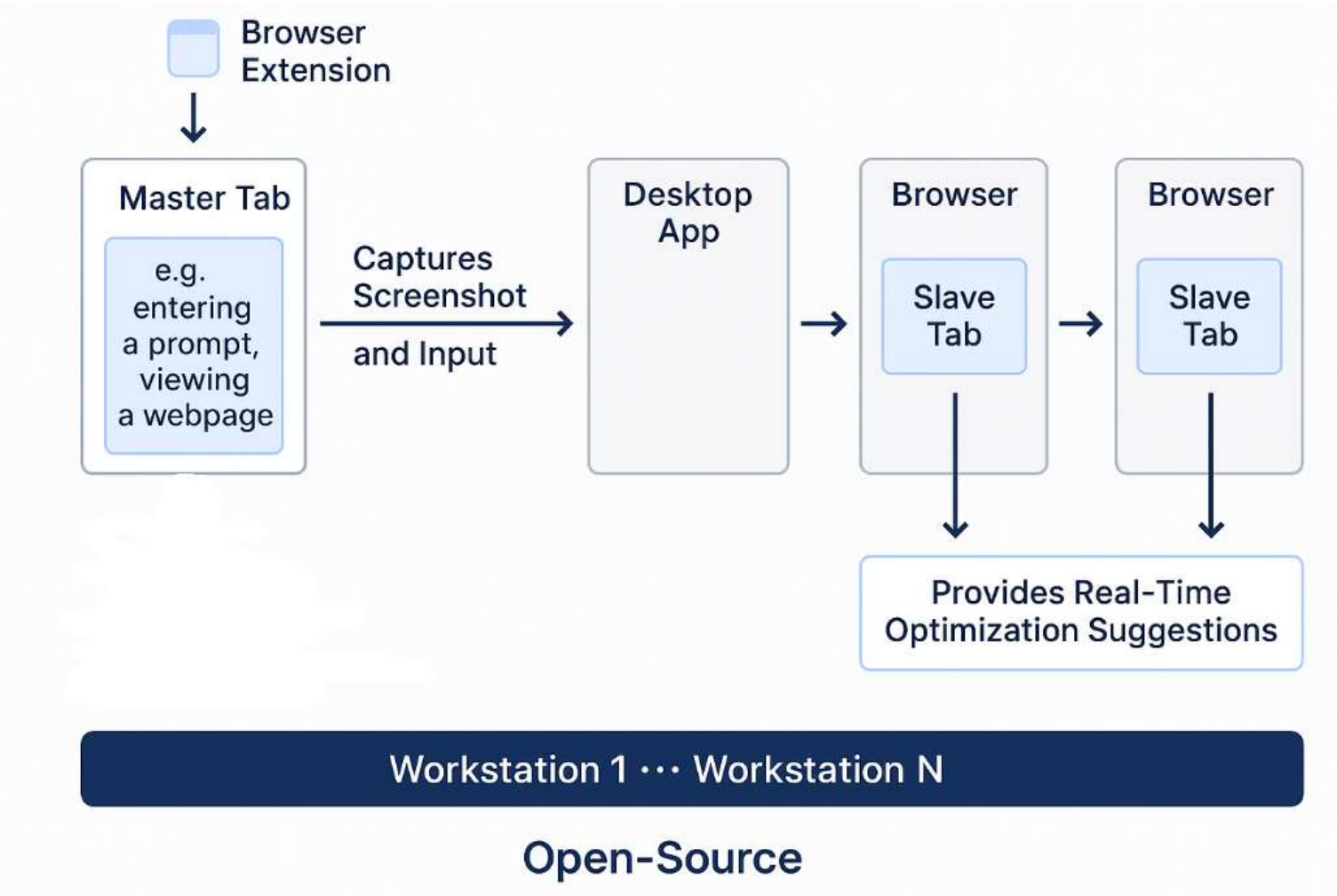
OpenGiraffe – Open-Source Browser-Based AI Agent Orchestration Framework



Paper by

optimando.ai

Abstract We introduce a novel open-source 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 **open-source** 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.

The architecture with the name **OpenGiraffe**, uses a lightweight browser extension to connect browser tabs to a local orchestrator desktop application. Users assign roles to tabs as either **master interfaces**—the primary point of interaction—or **helper agents**, which are implemented as browser tabs. To support repeatable and scalable workflows, the system allows entire browser sessions, including tab roles and orchestration logic, to be saved and reloaded. This eliminates the need to reconfigure sessions manually. Additionally, the orchestrator includes built-in templates for intent recognition and optimization logic,

enabling the system to proactively interpret user goals and deliver relevant suggestions without manual prompting. The AI agents in the helper tabs follow simple instructions written in plain language—no coding needed.

This is a foundational design choice: All helper agents operate within the browser and run dedicated AI models—such as ChatGPT, Claude, DeepSeek, Gemini, Mistral, Llama, Grok, or even autonomous systems like Google's Project Mariner. These models can be integrated in various ways: via APIs into a custom website (enabling community-driven templates), through the providers' web interfaces, or by hosting open-source models locally—typically exposed through a browser-based UI as well.

In the default OpenGiraffe architecture, each AI helper and AI coordinator agent is hosted in its own browser tab. This setup is essential when working with web-based LLM chatbots, where the model interface is embedded in the frontend of a website and cannot be accessed via direct API calls. In such cases, the browser orchestrator must manage real tabs to maintain context and simulate user interaction.

However, when LLMs are integrated via APIs—whether from cloud providers or locally hosted models—it becomes technically feasible to consolidate all agent logic within a single custom-built website, running in a single browser tab. This orchestration page is hosted locally on-premise and defines each AI helper and coordinator agent using structured HTML snippets with embedded metadata (e.g., unique IDs, roles, endpoints, reasoning logic, and parameters). The browser extension then reads and executes the orchestration logic directly from this unified interface.

The unified page supports multiple profiles and modular AI agent configurations, each of which can be independently toggled—enabling flexible activation of specific workflows or toolchains. Hosting the

orchestration layer on-premise not only enhances privacy and data sovereignty, but also enables safe, community-driven sharing of reusable agent templates without exposing backend infrastructure.

Functionality remains consistent: master input tab context is preserved, and agent outputs are routed to display slots—while the traditional "one tab per agent" abstraction becomes optional in API-driven environments.

🔁 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. Browser-Tab Mode (**OpenGiraffe:** Default Web-Based Architecture)

Each Helper Agent is launched in a dedicated browser tab, managed by a browser-based orchestrator. This setup is essential when working with web-based LLM frontends (e.g., ChatGPT, Claude) where API access is unavailable. The orchestrator simulates user interaction across tabs to maintain context and autonomy for each agent.

2. API-Based Mode (Custom Website Integration)

When LLMs are accessible via API, all Helper Agents can run inside a single custom-built web interface. Each agent is defined as a modular component (e.g., HTML + metadata) with a unique ID. The browser extension routes queries directly to the appropriate module within the page—allowing agent logic to execute concurrently without multiple browser tabs.

Regardless of the mode, Helper Agents may utilize:

- Structured memory from a local or embedded vector database
- Context-specific data filters (e.g., safety documents only)
- Agent-specific prompt engineering or tool access

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 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 **OpenGiraffe** 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 HTML interface** by integrating more complex reasoning steps, business rules, or specialized input fields.*
- *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 **OpenGiraffe** 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 **OpenGiraffe** orchestration system is n8n — a source-available, browser-based, and locally hostable automation engine. While not required for basic operation, it extends the system's capabilities significantly by enabling deeper backend logic, integration with third-party tools, and automation of multi-step workflows.

n8n integrates cleanly into this modular OpenGiraffe 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 intents 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 **OpenGiraffe** 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.

OpenGiraffe 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, **OpenGiraffe** 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, **OpenGiraffe** 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 **OpenGiraffe** 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 OpenGiraffe, 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 **OpenGiraffe** 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 **Optimando'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.

📷 **OpenGiraffe & 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 **OpenGiraffe**-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 OpenGiraffe 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, **OpenGiraffe** 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
OpenGiraffe never auto-submits forms. If a QR 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"
  - "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 **OpenGiraffe** 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, 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 OpenGiraffe 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 OpenGiraffe or any of its services.**
✅ **All sensitive operations occur locally and client-side, maintaining strict privacy by design.**

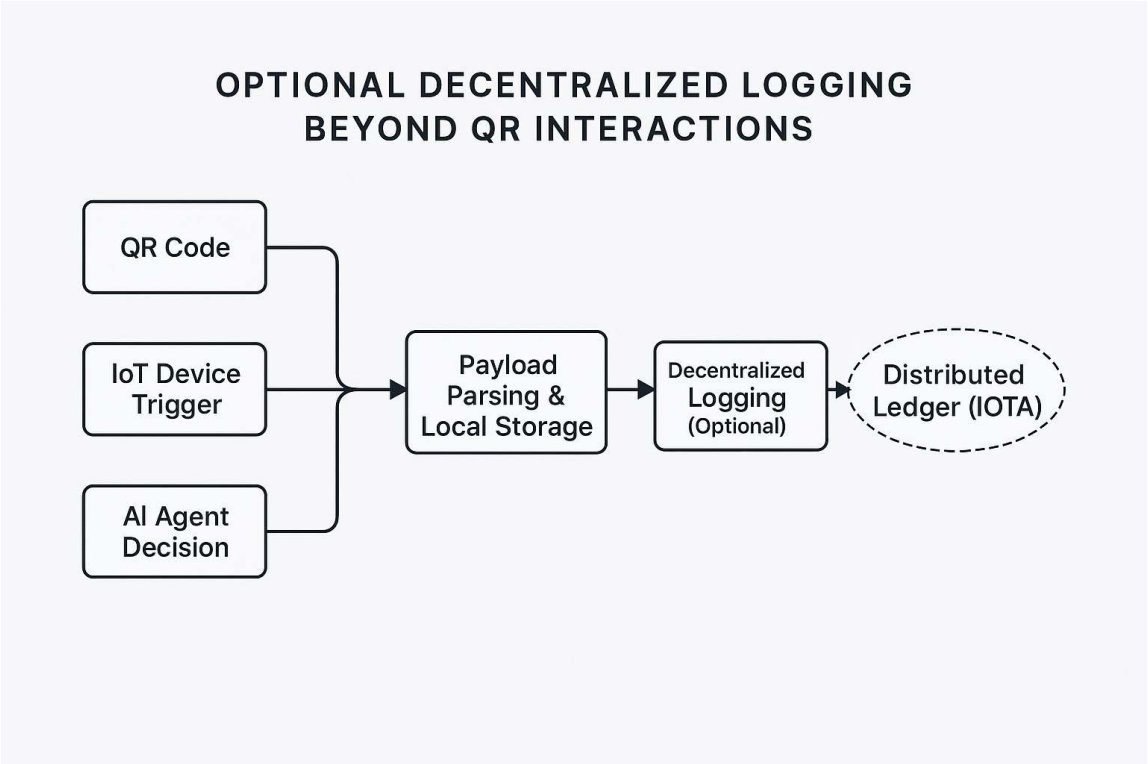
Extendable to Other Domains Beyond QR

This decentralized verification approach is not limited to QR interactions.

→ It can also be applied to other components of the **OpenGiraffe** 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.**



🔗 WR Code Security and Local-First Orchestration in OpenGiraffe

To enhance security, integrity, and privacy in AI-driven automations, the **OpenGiraffe** 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, **OpenGiraffe'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 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 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 PoO mechanism guarantees 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 PoO-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., OpenGiraffe), without compromising the **trustless, transparent execution model of WRCode templates**.

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

- Edit templates
- 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 PoO 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, orchestrated user experiences—intelligent by design, controlled by you.

🔑 Geofence-Orchestrated Intelligence

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

OpenGiraffe 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, **OpenGiraffe** 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, **OpenGiraffe** 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.

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.**

📄 OpenGiraffe & 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 **OpenGiraffe** 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 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 open-source 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 + prompts for piece count/status → generates instant report	Saves time, reduces errors, offline-capable
📝 Legal Document Automation	WR on contract triggers automation that pulls in stored company/user PII → asks for missing fields (e.g., delivery date) → outputs ready-to-sign doc	Accelerates legal/admin processes while ensuring compliance
🎁 Coupon & Offer Claims	Marketing flyer with WR triggers personalized coupon claim → automation uses stored user data + asks for optional preferences → generates offer	Drives engagement without forcing app installs or logins

🚧 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	Boosts security, reduces friction, maintains privacy
🚗 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	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, **OpenGiraffe** 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 OpenGiraffe (QR-Giraffe Module)

1. Objective

OpenGiraffe 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 is securely verified through an cryptographic hash.
- Templates 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.
- Every WR Code and automation template is securely verified through an Solana-based hash.
- Templates without a valid signature or with altered structure are strictly not executed.

- 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. OpenGiraffe 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, **OpenGiraffe** 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. This standard operates independently of the **OpenGiraffe** software, which remains fully **open source** and is not affected by any validation or governance process. 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 consistency**, and **issuer accountability**.

To support integrity, trust, and transparency, wrcode.org serves as a public validation platform where organizations can voluntarily authenticate themselves and register their QR automation templates. This process allows templates 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, while companies can opt into higher validation standards to build trust and visibility. Participation in the validation process is voluntary, ensuring that the decentralized and open character of the WR Code standard is preserved.

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 OpenGiraffe.

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

OpenGiraffe 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 alter the technical structure of QR Codes but introduce a new way to use standard QR Codes as triggers for verifiable, flexible automations. This enables any individual, developer, or organization to create and use WR Codes entirely within private or internal infrastructures, without mandatory registration or licensing. This ensures fast, cost-effective automation while maintaining full privacy and 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.
- 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 OpenGiraffe 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 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** , hosted transparently
- **Cryptographic runtime validation** , anchored on **Solana**
- A user-controlled **WRVault** for secure, privacy-preserving personalization

- And a strict **governance model** that enforces **publisher verification**, **template integrity**, and **policy compliance**

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 and policy manifests** that are publicly **hosted on wrcode.org**, versioned, and **tamper-proofed** through WRStamp signatures.
- **Anchoring on Solana**, 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 guarantees 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 Automation Templates:** Every automation template must be cryptographically hashed and anchored on the **Solana blockchain** to guarantee immutability and resistance to tampering.

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:** Vault applications responsible for handling sensitive data must:

    - Adhere strictly to **wrcode.org** guidelines.

    - Be published under the **GNU Affero General Public License v3 (AGPLv3)**, the same license used by the OpenGiraffe project.

- Be cryptographically hashed.
- Be anchored on the **Solana blockchain** to ensure authenticity, traceability, and tamper-proof verification.

The Vault must operate as a **standalone, independent application** to guarantee that all sensitive actions, including PII handling and automation approvals, occur within an untampered, auditable environment. All **whitelisting of WRCode providers** and **user consent decisions** must be made exclusively within this original Vault application, ensuring that no proprietary software can override or simulate user approval.

To prevent unauthorized duplication or misuse, each Vault instance must be **cryptographically bound to the device on which it is installed** . This is achieved by generating a **device-specific hardware fingerprint** derived from secure elements such as a Trusted Platform Module (TPM), Secure Enclave, or equivalent. The hardware fingerprint, combined with the Vault software hash, is **anchored on the Solana blockchain** to establish a tamper-proof link between the Vault and the device.

At runtime, the Vault must verify its own software integrity and confirm the device's hardware fingerprint before permitting any sensitive operation or data access. This ensures that even if Vault software is copied or tampered with, PII remains inaccessible without the original, trusted hardware environment.

To maintain frictionless user experiences for trusted services, the Vault must allow users to **explicitly approve initial PII access** and optionally **whitelist specific WRCode providers** for streamlined future interactions. This whitelisting remains entirely under user control, with the ability to review, revoke, or modify permissions at any time.

To maintain frictionless user experiences for trusted services, the Vault must allow users to **explicitly approve initial PII access** and optionally **whitelist specific WRCode providers** for streamlined future interactions. This whitelisting remains entirely under user control, with the ability to review, revoke, or modify permissions at any time. Access to PII is only granted if:

1. The proprietary software calling the Vault passes integrity checks.
2. The Vault software matches the original, hashed version anchored on Solana.
3. The user has explicitly granted or whitelisted the specific provider.

Vaults may optionally log user approvals and whitelisting events locally in a tamper-proof way to enhance transparency, user control, or compliance needs. These logs are strictly informational and do not contribute to the core security of the **WRCode** framework, which is fully enforced through cryptographic proofs, anchored software integrity, and device-bound Vault verification. Proprietary **WRCode** apps must automatically anchor the hashes of every new software release on **Solana** to

maintain verifiable software integrity. This ensures that all participants follow the guidelines strictly and that users can trust the process without compromise.

Only Vaults issued by verified **WRCode** publishers and listed on **wrcode.org** are considered compliant and safe for use.

Together, these elements establish a comprehensive security model that protects users from fraud, manipulation, and unauthorized data access.

Unregistered automations are strongly discouraged, as they expose users to significant risks, including:

- Use of unverified Vault software that may mishandle or leak sensitive data.
- Execution of altered or insecure automation templates.
- Absence of verifiable issuer identity or template integrity.

Only **WRCode**-compliant automations—fully registered, verifiable, and tamper-proof—should be trusted for customer-facing or security-sensitive applications.

Consumers and enterprises must:

- Use only **Vaults published by verified WRCode issuers** listed on **wrcode.org**.
- Verify automation templates through the public registry before execution.

To maintain openness and flexibility, any organization may issue its own Vault software, provided that the software is **fully open-source, published on wrcode.org, auditable, and anchored tamper-proof on Solana**.

Such Vault applications must:

- Be published under an approved open-source license.
- Follow the strict security and privacy guidelines set by **wrcode.org**.
- Be cryptographically hashed and anchored on **Solana** to ensure authenticity, integrity, and verifiability.

This approach allows companies to build their own compliant Vault solutions while maintaining the core trust and security guarantees of the **WRCode** ecosystem.

This framework prevents fraudulent actors from manipulating automations or distributing unauthorized Vaults.

Future Execution Layer: **QRGiraffe**

The forthcoming **QRGiraffe** orchestration environment will leverage the full potential of the **OpenGiraffe 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 guarantees 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
- 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 **OpenGiraffe 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 **OpenGiraffe 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 Identity for **WRCode** Issuers: All entities generating **WRCode** automations—including merchants, service providers, and organizations—must register through wrcode.org and undergo identity verification. Their identities are anchored tamper-proof on the Solana blockchain as part of **WRCode's** foundational security design.
- Free Access for Users: Individual users interacting with **WRCode** automations retain the freedom to do so without mandatory identification. Identity anchoring for users is strictly optional and only applies where specific use cases (e.g., regulatory compliance, service personalization, or recovery) require it.

Optional Identity Anchoring for Users

Users may opt into a lightweight identity anchoring system to:

- 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.

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

---

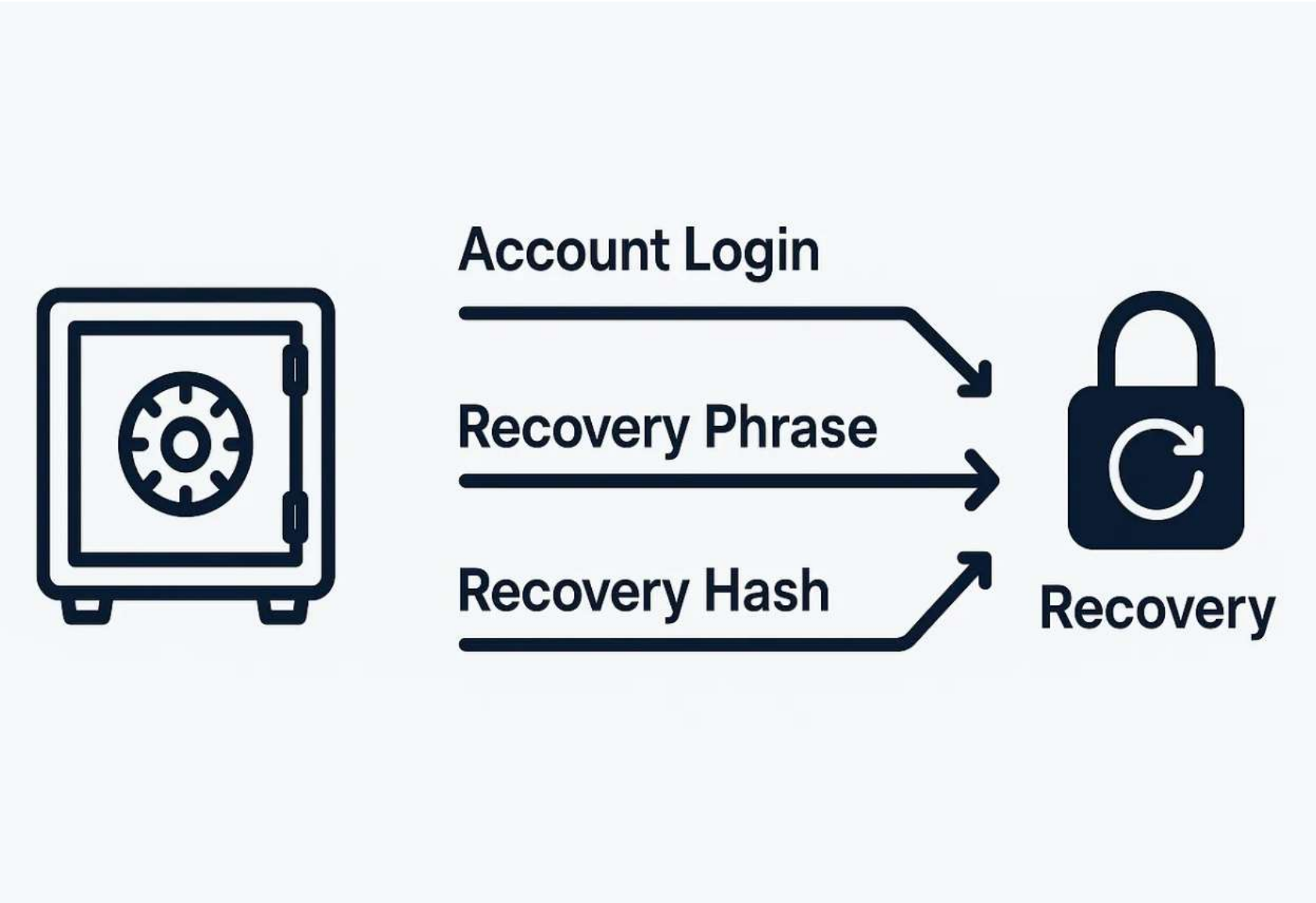
WRPass — Decentralized Identity & Access

WRPass replaces conventional logins and verification methods with a secure, WR-based identity layer. Acting as an on-premise, **open-source** 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 store 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.
    - 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:
    - Business logic is **not executed or interpreted** at this stage. The vault performs only validation and orchestration setup based on publicly anchored instructions.
    - 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 wrcode.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 and templates 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 files and templates 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 guarantees:

- 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

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 **OpenGiraffe** 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 user scans a **WRCode** at a bus or train station.

- Local AI asks: *"Where are you going?"*

- Vault contributes personal preferences like preferred walking distance, step-free routes, or visual assistance.
- Geofence confirms the user is physically at the station, enabling precise, location-aware guidance.

Impact: Transit providers benefit from reduced app maintenance, lower onboarding barriers, and increased inclusivity — especially for elderly users or tourists unfamiliar with the system. Since only local context is used and no personal data is shared, the **WRCode**-based interaction is fully anonymous.

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 neccessary 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

guarantees 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 guarantees 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 OpenGiraffe: 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 OpenGiraffe 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 guarantees 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 OpenGiraffe, the browser-based AI orchestrator developed by Optimando.ai.

⚙️ Why OpenGiraffe 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 OpenGiraffe 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 OpenGiraffe, 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

OpenGiraffe 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, OpenGiraffe 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, OpenGiraffe 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 only—unregistered **WR Codes** or AI templates 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, open source, 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 guarantees. 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 OpenGiraffe. 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 OpenGiraffe, 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 **OpenGiraffe** 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 **OpenGiraffe** 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 → **OpenGiraffe** 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.

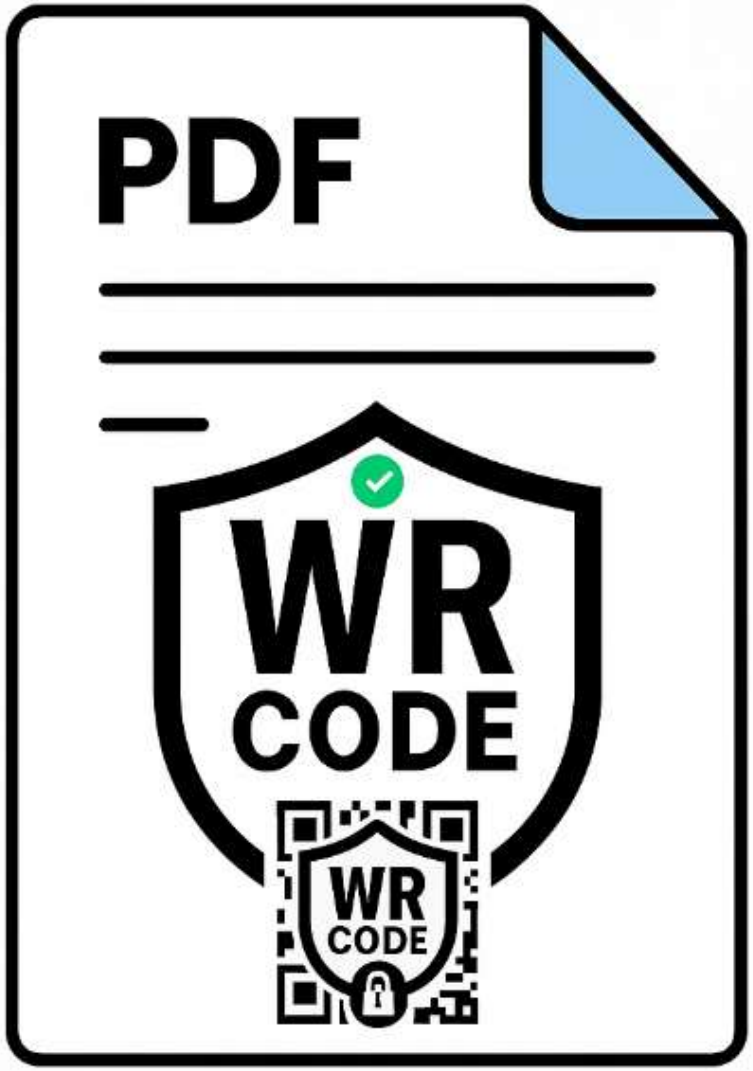
🧠 In Summary

WR Codes are the ignition key. **OpenGiraffe** is the engine. WR Codes are already powerful when scanned on mobile devices—but they become truly magical when connected to **OpenGiraffe**. 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 **OpenGiraffe** 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



WRStamped : A Trustless Protocol for Verifiable Document Integrity

Abstract: This paper proposes a vendor-neutral, trustless protocol for generating and verifying digitally anchored PDF documents. The system enables individuals and organizations to produce locally generated PDF files embedded with **WRCode** identifiers—compact QR codes encoding a document hash, template signature, and account-bound sender identity. These identifiers are cryptographically verified using Merkle-tree hashing and publicly anchored on high-performance blockchains such as Solana or Optimism. Optional batch-level proofs may also be submitted to archival networks such as Solana. Unlike PKI-based signature systems, **WRCode**-stamped PDFs provide privacy-preserving verifiability without exposing user identity or relying on certificate authorities, proprietary platforms, or cloud infrastructure.

1. Introduction Email-based fraud, invoice manipulation, and unverifiable PDF generation pipelines represent growing threats to secure digital communication. Current approaches to digital signatures are either too complex for widespread adoption, locked into proprietary ecosystems, or fail to provide a simple, visible proof of authenticity that users can verify independently.

We propose **WRCode** Stamped PDFs: a lightweight, open-source protocol that embeds a **WRCode** QR block into the document footer. This **WRCode** encodes a signed reference to the document's content hash, tool/template used, and the originating **WRCode** identity, all anchored to a distributed

ledger (e.g., Solana, Optimism, IOTA). The result is a self-contained, auditable document format that supports offline validation, local generation, and regulatory transparency.

2. System Overview

Each **WRCode** Stamped PDF is created through a controlled rendering tool (e.g., LibreOffice headless, LaTeX, or WeasyPrint) that ensures deterministic output. The **WRCode** QR block is added during or after rendering and encodes:

- SHA256(PDF document content)
- Template/tool version identifier
- Sender identity bound to a verified account at wrcode.org
- Timestamp of generation
- Solana reference (transaction hash or Merkle root)

Templates are defined by their visual structure and tool version, and must be transparent, open source, and auditable. They are published on WRCode.org and may be forked under open terms, but must be registered and anchored on WRCode.org before use. Anchors are created through Solana messages containing a hash of the document metadata bundle. This requirement ensures that all templates used in document generation are tamper-proof, transparently verifiable, and auditable. By

mandating registration and anchoring of templates, **WRCode** Stamped PDFs operate under a zero-trust assumption: no rendering tool or layout logic is implicitly trusted unless independently verifiable via WRCode.org.

3. Security Properties

- Integrity: The document's content hash is tamper-evident and verifiable.
- Origin Assurance: Sender identity is cryptographically linked to a registered wrcode.org account.
- Anchor Immutability: Anchored hashes on Solana cannot be altered or deleted. No personally identifiable information (PII) is shared or leaked in this process—only cryptographic hashes of the document and related metadata are stored.
- Online Verifiability: The WRCode can be scanned from print or screen and validated using a WRCode scanner with an active internet connection. Real-time verification is handled by local PostgreSQL-based integrity checks combined with Merkle tree structures. All relevant proofs are immutably stored and periodically batched for anchoring on performant public blockchains such as Solana or Optimism. Additionally, anchoring to Solana provides long-term, tamper-evident integrity for archival purposes. Blockchain anchoring is not optional but forms the

foundation for traceable, cryptographic auditability. All components are open-source and locally executable.

4. Applications

- Invoice & Payment Authenticity: Proof that an invoice is genuine and unaltered.
- Regulatory Compliance: Financial, legal, or HR documents with verifiable origin.
- AI-Generated Document Trust: Allows AI-generated outputs to be anchored and traced.
- Automation Triggers: WRCode only functions online and requires an active internet connection to verify anchors and registered templates. It can act as an authenticated trigger for scanning agents or connected devices once verification is complete. can act as an authenticated trigger for scanning agents or devices.

5. Comparison to Existing Approaches

Feature	WRStamped PDFs	PKI Signatures	Cloud Signing Platforms
Local generation	Yes	Partial	No
Offline validation	Yes	Often no	No
Vendor lock-in	No	Yes	Yes
Human-readable WR	Yes	No	Partial
Identity privacy	Account-bound, selectively disclosed	Always exposed	Always exposed

---

6. Implementation & Toolchain

WRCode Stamped PDFs can be created using a signed local binary (e.g., wrstamp-tool) that integrates:

- Deterministic PDF generation (LibreOffice, LaTeX, etc.)
- WRCode embedding

- SHA256 hashing

Blockchain anchoring (e.g., Solana, Optimism) and optional IOTA archival submission. Proofs are structured via Merkle trees and batched before anchoring

Verification tools include:

- Mobile and desktop QR scanner apps
- Browser-based WRCode checkers
- Vault-integrated agents within orchestration systems like OpenGiraffe
- Zero-knowledge identity binding to WRCode hashes
- Selective disclosure of sender metadata
- Interoperability with Verifiable Credentials and EU eIDAS 2.0
- Formal **WRCode** Template Specification & governance via **wrcode.org**

Conclusion: WRCode Stamped PDFs present a novel composition of well-understood cryptographic primitives and distributed anchoring techniques, arranged to meet real-world demands for verifiable, privacy-preserving document automation. While components like QR-based hash encoding and cryptographic-based anchoring have precedent individually, their integration into a trustless protocol combining deterministic document generation, auditable template registration, identity-bound WRCode issuance, and protocol-conformant scanning represents a previously undocumented solution architecture.

This work is released as an open-source invention disclosure. It is not a formal patent application, and the author makes no exclusivity claims. While every effort has been made to verify originality at the time of publication, the author explicitly disclaims legal warranties of novelty or non-infringement. Readers, implementers, and derivative authors must perform their own due diligence before reuse, especially in regulatory or commercial contexts. Importantly, WRCode extends beyond static document verification. It also acts as a secure automation trigger for downstream AI agents—unlocking verifiable, automated workflows such as invoice parsing, bookkeeping, or compliance auditing. This convergence of verification and automation adds a uniquely powerful dimension not found in existing PDF signing ecosystems.

By openly publishing this system and binding it to a ledger-verifiable governance model, WRCode establishes a foundation for interoperable, tamper-evident documentation standards—designed for human readability, machine validation, and scalable trust without reliance on vendor-specific signing

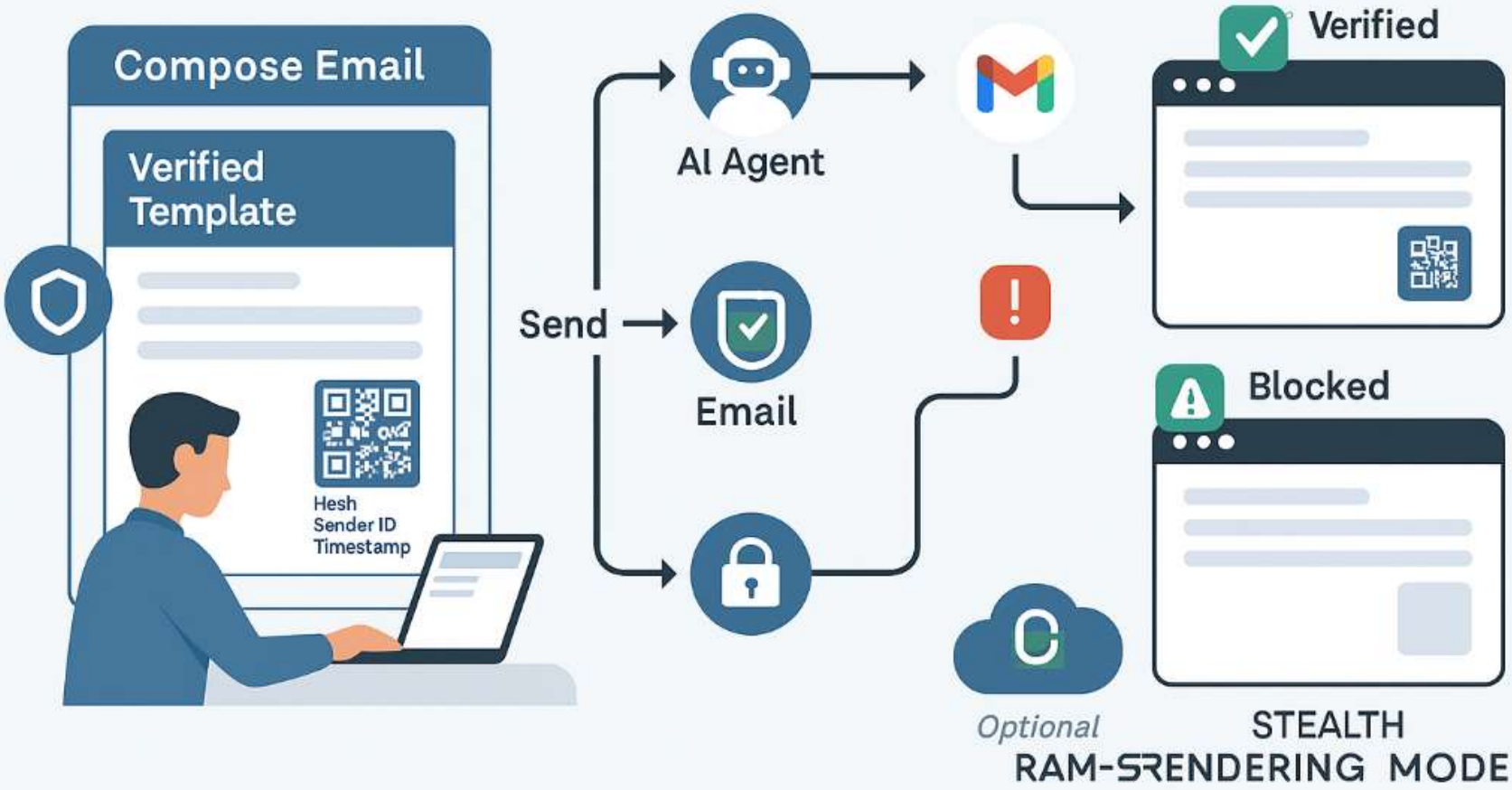
infrastructure. This publication is timestamped to serve as a public contribution to the open-source and academic community. All described mechanisms and templates will be released under an attribution-permissive license via wrcode.org.

Additionally, the WRCode protocol supports advanced AI-driven automation workflows. Once a WRStamped PDF is verified, additional AI agents can be triggered to perform deep invoice analysis or initiate fully automated bookkeeping processes. This extends the protocol beyond passive verification, enabling a trustless automation layer anchored in verifiable document integrity.

WRCode Stamped PDFs enable a new class of verifiable documents, combining privacy, open standards, and ledger-based proof of integrity. By anchoring document metadata and hash in a distributed ledger and embedding the resulting proof as a WRCode QR, this protocol provides scalable, user-friendly trust infrastructure for both digital and paper-based workflows.

This publication serves as an initial disclosure and timestamped record of the invention. The author explicitly disclaims any warranty of novelty or non-infringement. This work is provided purely for the benefit of the open-source and academic community, and no exclusivity or intellectual property claims are asserted. Readers and implementers are advised to conduct their own due diligence before reuse or deployment in regulated or commercial environments. All protocols, structures, and implementation logic described herein will be released under an open, attribution-permissive license and further documented at wrcode.org.

SecureEmail Module in OpenGiraffe Orchestrator



SecureEmail Module – Enforcing Trusted Email Composition and Rendering with WRCode

1. Introduction

The SecureEmail Module is a security extension within the OpenGiraffe Orchestrator, designed to prevent phishing, spoofed identities, and tampered emails. It enforces WRCode-anchored, auditable templates and attachments designed to prevent phishing, spoofed identities, and tampered emails. It enforces WRCode-anchored, auditable templates and attachments via a zero-trust security model. Integrity is enforced using Merkle-based validation, local tamper-proof storage (PostgreSQL, MDIKS), and blockchain anchoring. Public anchors are submitted to performant blockchains like Solana and Optimism. IOTA provides an additional archival integrity layer for long-term traceability.

2. How It Works

- Only verified WRCode users can log into the composer via WRCode scan, initiating a session that cryptographically links email generation to their WRCode.org account.
- These are published on WRCode.org and anchored on high-performance blockchains such as Solana or Optimism for real-time and audit-grade verification. Optionally, a batched Merkle-root may be submitted to IOTA for long-term archival purposes only.
- **WRStamped Email Sending Process**
- When the user clicks "Send," the following steps occur:

- 🔐 **Hashing**
- The entire email body and all attachments are hashed together to create a single canonical digest.
- 📄 **The resulting hash is:**
- 📎 Anchored on **Solana or Optimism** for immutable, timestamped integrity verification.
- 📋 Embedded in the email **footer as a WRCode** (visual representation and manual verification or automation trigger).
- 📬 Included as an **X-Header** for automatic detection and validation by WR-aware email clients.
- 🧷 Used to validate **WRStamped attachments** , ensuring they match the original content.
- 🌍 Combined with the **geographic region of the sender** and included in the WRStamp context.
- 🔑 The **recipient's public key** is automatically retrieved from a WRGuard-verified record, linked to the recipient's email identity. The message is optionally encrypted using this key, ensuring that only the intended recipient can decrypt it. No manual key exchange is required. Encryption is recommended for sensitive data, but not required for WRStamped email participation. However, secure opening (e.g., in a sandboxed orchestrator container) is mandatory for interacting with critical templates (e.g., forms, payment instructions). The sender's proof of

origin is derived from a locally resolved, coarse-grained location signal (e.g., IP or mobile network). A cryptographic algorithm converts this into a verifiable statement about the sender's broader region (e.g., "Western Europe" or "North America"), without referencing city-level or exact locations. This regional statement is then anchored cryptographically as a zero-knowledge token — making it publicly verifiable while preserving sender anonymity. Even wrcode.org has no access to the raw geographic data. Only a **country-level statement** (e.g., "Germany", "France") is mathematically derived from the local origin data and publicly anchored cryptographically. While the region claim is verifiable by anyone, it is coarse enough to preserve user anonymity. This strikes a balance: preventing false claims about geographic origin without enabling individual tracking or precise localization. This mechanism protects privacy while preventing false geographic claims, helping to reduce phishing and impersonation risks.

The email composer in the secure layer of the **OpenGiraffe** orchestrator injects this verified content into supported providers like Gmail or Outlook using DOM manipulation.

✨ **Enhancements:**

- 📄 **Structured WRStamped Manifest**
- A WRStamped manifest is optionally attached to describe included file types, the scope of change, automation tags, and optional expiration or access rules.

- ⚙️ **Symmetric Key Exchange for Performance**
- For large payloads, a symmetric encryption key (e.g., AES-256) is generated locally and used to encrypt the entire email content and attachments. To maintain end-to-end encryption (E2EE), this symmetric key is then encrypted with the recipient's public key. The recipient first decrypts the symmetric key using their private key, then uses it to decrypt the message content. This hybrid approach ensures strong encryption while remaining computationally efficient and scalable for large data transfers.
- 🚫 **Forwarding Controls**
- Optionally include WRPolicy metadata for:
- 📤 Forwarding permissions
- 🔒 Access limits
- 🚨 Tamper detection routing (e.g., notify orchestrator or WRDev account upon anomalies)

✅ This ensures emails are not only secure and verifiable, but also orchestratable in real-time across WR-aware systems.

🔐 Secure Layer: Optional for Emails, Built-in for Automations

The use of the **Secure Layer** is entirely **optional for viewing emails but for the security features they need to be opened in the secure layer of OpenGiraffe.**

All WRCode-triggered automations require a **WRScan**, which enforces integrity and origin checks by design. This makes automation workflows **inherently secure and tamper-proof** , even **outside** the Secure Layer. No unsafe automation can be executed without passing verification.

In contrast, **WRStamped emails** can be viewed **outside** the Secure Layer, but only in a **passive mode**. To unlock **full protection** — such as:

- Preventing tampered emails from being opened,
- Enforcing origin checks,
- Enabling trusted autofill or **drafting** AI-based reply generation without even scanning the WRCode,

...the email must be opened **within the Secure Layer** .

Emails that **fail WRStamp verification** cannot be opened at all within the Secure Layer of **OpenGiraffe** . This ensures that only **authentic and untampered messages** reach the user in secure contexts.

This architecture allows broad compatibility while enforcing **trust boundaries where it matters most** — making the system practical, adoptable, and secure by design.

3. Inbox Protection and Pre-Rendering Validation

- When an inbox is opened in a tab, the SecureEmail Module performs pre-rendering validation (no opening possible until verified):
  - Hashes each message body
  - Compares it to the cryptographically-anchored hash and X-Header
  - Blocks unverified messages before any rendering occurs
- If a message passes all checks, it becomes interactively clickable. If not, it remains locked with a visible warning.

4. Security Policy and Restrictions

- **Only WRStamped messages are rendered**
- **No external links allowed**
- **Only verified attachments are accepted**

This design dramatically reduces the risk of phishing, spoofing, and malicious payloads.

5. Stealth Rendering (Planned Feature)

A future enhancement will enable RAM-only containers for internet surfing:

- Zero local persistence
- Secure memory clearance after closure
- Ideal for whistleblowers, journalists, and sensitive communications

6. Impact on Spam and Risk Mitigation

The WRCode-based enforcement blocks all unauthorized email rendering, making traditional spam and phishing largely obsolete. This has particular value for high-risk individuals such as executives, journalists, and government officials. They gain a practical security layer that prevents social engineering by blocking unaudited messages at the source.

7. Example Workflow: AI-Guided Secure Communication

An OpenGiraffe user issues a voice command:

“NPC7455, write a reply to Arne confirming the invoice was received and schedule a follow-up.”

The orchestrator:

- Loads the correct WRCode-certified reply template
- Uses internal LLM agents to compose the response
- Anchors the output and injects a WRCode footer together with the body hash
- Sends the email through a connected Gmail session

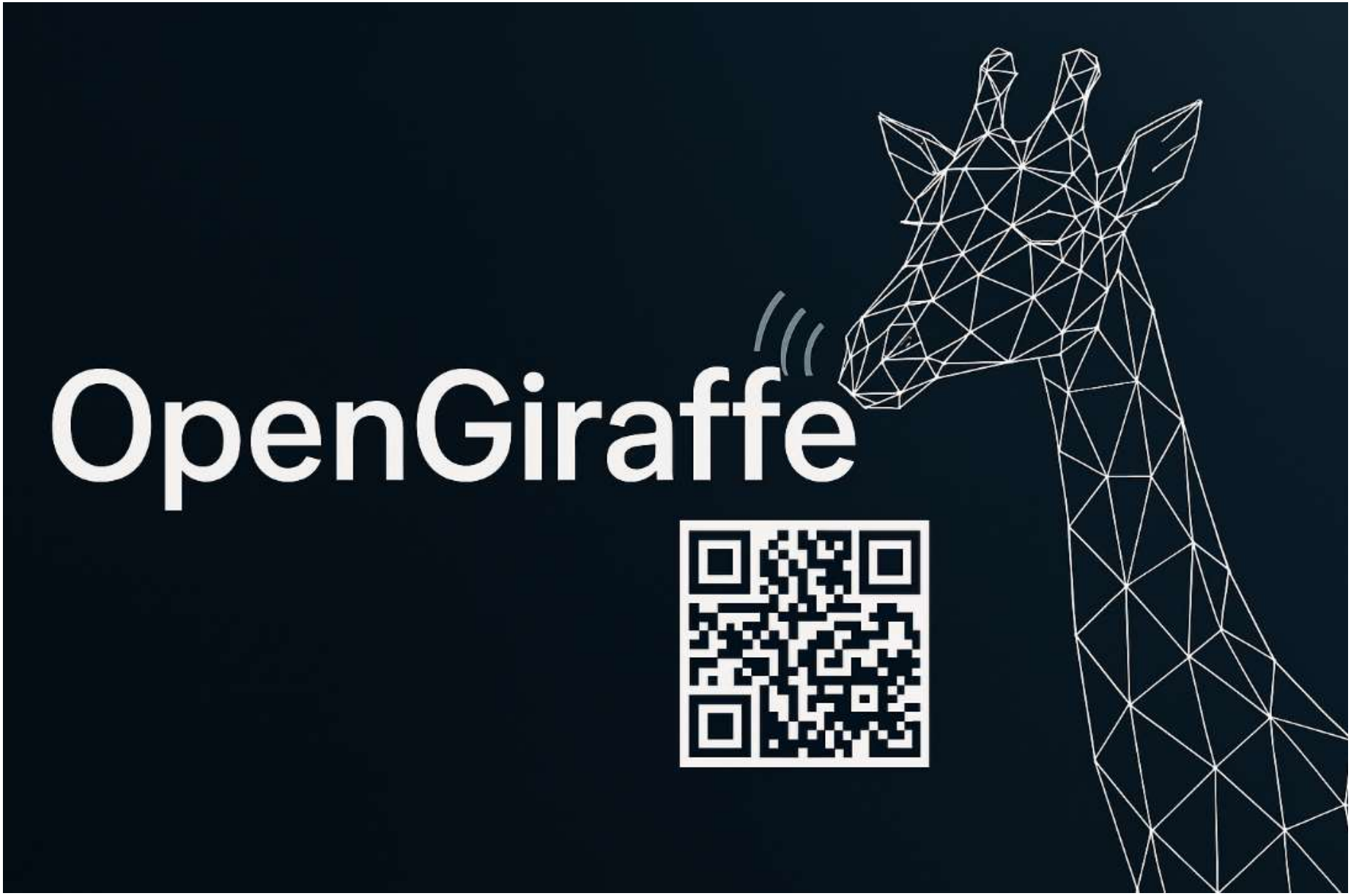
If the recipient is also running a WRCode-compatible orchestrator or browser extension, the email won’t render unless the WRCode footer is verified and the email matches the approved structure.

8. Conclusion

The SecureEmail Module transforms traditional email into a verifiable, tamper-proof, and automation-ready communication channel. By combining identity-bound templates, anchored metadata, and rendering-time enforcement, it ensures that only fully trusted emails are composed and displayed.

All templates are open source, auditable, and governed by WRCode.org. This makes the system not only secure but interoperable and scalable.

With its roadmap for stealth rendering and AI-powered composition, SecureEmail represents a key milestone toward universal, machine-verifiable trust in human communication.


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 open-source, 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 open-source 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 code libraries 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 Runtime 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 Runtime Enforcement Framework replaces that assumption with cryptographic trustless security.

✅ Every line of executable code must reside in a WRStamped file.
❌ If even a single line of running code is not WRStamped — the system halts.

📦 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	✅
Themes	✅
SQL execution paths	✅ (via Merkle-root trust chain)
Vault/decryption logic	✅
WordPress core overrides	✅
Custom PHP/JS code	✅
Shortcodes, handlers	✅

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()

Licensing is now bound directly to:

- A WRCode account or device identity
- A zero-knowledge proof (ZKP) that validates license state without revealing logic
- A WRStamped module that defines licensed behavior in advance

Need	Legacy Method	WRCode Approach
Verify license	eval() or dynamic callbacks	ZKP bound to WRCode account or device
Tiered access	Inline PHP switches or remote API	Pre-stamped modules with license scope
Trial expiry	Obfuscated timestamp logic	Enforced through time -bound WRStamp policy
Piracy protection	Obfuscated logic injection	Immutable, tamper-proof execution paths

The result:

No runtime tampering

- No secret logic injection
- No trust-by-obscurity
- No open-source requirement

🔐 Proprietary Code Is Still Welcome

WRCode does not require that your code be public.
You can deliver closed-source, obfuscated modules — as long as they are:

- Pre-compiled
- Cryptographically WRStamped
- Immutable at runtime

You keep your IP. But you lose the ability to inject or modify logic after deployment.

This isn't about transparency. It's about verifiability and user safety.

📦 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
- 🔐 The system enters fail-closed lockdown

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	✅ Executes
File is missing/modified	❌ Execution denied
Plugin generates code at runtime	❌ Blocked unless pre-WRStamped
Plugin uses eval/license logic	❌ Incompatible (use WRCode API/SDK)
Plugin uses signed static modules	✅ Fully compliant
Query called from trusted stack	✅ Executes
Query called from unknown origin	❌ 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.*

No WRStamp. No Execution. Trustless Security
That's the new standard.

🔐 WRGuard for WordPress- Technical Flow

🗂 Outline

1. WRStamped Core Hashing – Ensure the WordPress core and critical runtime files are immutable and verifiable.
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. Encrypted Credentials Vault – Database access credentials are encrypted and only unlocked on successful verification.
4. index.php Guard Layer – The WordPress boot process is halted unless all integrity checks pass.
5. WRGuard Dev accounts: Developer Toolkit for Plugin WRStamping – Allow plugin authors to generate signed WRStamps for secure distribution.

🛠 WRGuard Dev Accounts: Developer Toolkit for Plugin WRStamping

To open a WRGuard Developer Account, plugin authors must complete a domain-based proof-of-origin verification. This process ensures that only verified domain owners — those who control a valid and traceable web property — can generate and attach WRStamps to their distributed software (e.g., WordPress plugins).

This binding of code to a verifiable domain origin helps establish a trusted and auditable software supply chain, preventing anonymous or spoofed plugin distributions.

Importantly, the proof-of-origin process is explicitly designed to preserve privacy. It does not track or require the developer’s exact location or identity. Instead, the verification only establishes a country-level origin, such as the country in which the domain is registered or operated.

This country-based origin metadata is cryptographically attached to the WRStamp to allow recipients to validate authenticity without exposing any personal or precise geolocation data. The system is fully compliant with modern data protection regulations (e.g., GDPR) and reflects WRGuard's commitment to privacy-first, trustless security design.

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.

🔄 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.

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/)
- Critical runtime files like index.php, wp-config.php, mu-plugins/
- WRStamped plugin 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 guarantees 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:

- WRGuard::check_integrity() returns false
- index.php immediately stops execution with an HTTP 503
- db.enc remains encrypted and never decrypted
- 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

OpenGiraffe
WR
GUARD

🛡️ WRGuard: Technical Overview of Tamper-Proof Plugin & Theme Integrity in WordPress

WRGuard is a security and verification system that enforces real-time, tamper-proof code integrity across the entire WordPress software 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.

- 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 (roof).
  - Files are grouped under their plugin/theme/app or WP core (top roof).
  - The root hash of this Merkle Tree (the roof hash) is generated.

- The Merkle root is WRStamped and cryptographically anchored on public ledgers such as Solana, Optimism, and optionally IOTA, selected based on the required balance of speed, auditability, and permanence.WRStamping Protocol
- 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 ("roof").
- 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 templatemapdynamic WRCode for WRGuard Dev Edition.
- This templatemap 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
  - RAM process kill: All PHP-FPM or related memory-bound execution threads are forcibly terminated upon detection of tampering. 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 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 a full-stack integrity check within a dedicated Docker container, optimized for real-time monitoring. 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 httpdocs/ 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 guarantee integrity if executed from 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 OpenGiraffe 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 OpenGiraffe 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 (roof token) of the active WRStamped code stack.

Each WRStamped code block responsible for a database action automatically includes this token, which is then embedded as metadata into the corresponding write. To enable this, the respective

MySQL tables are automatically extended during WRGuard installation to include token-tracking fields (e.g., wr_token, wr_origin, or wr_signed_at), 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 token, the system logs the event and triggers a customizable audit workflow. This includes automatic diff generation, token validation, and rollback preparation. Unauthorized or suspicious writes are not rejected immediately but are overwritten or reverted shortly afterward using a background shell script bound to PostgreSQL. This script, configured per deployment, functions as a real-time "firefighter"—restoring the last verified state or quarantining modified records within milliseconds of detection.

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.

🔐 Pre-Execution Blocking Mode for High-Assurance Environments

WRGuard's default runtime enforcement 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.

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 unauditable. 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 OpenGiraffe 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.

WR CODE

🔐 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.

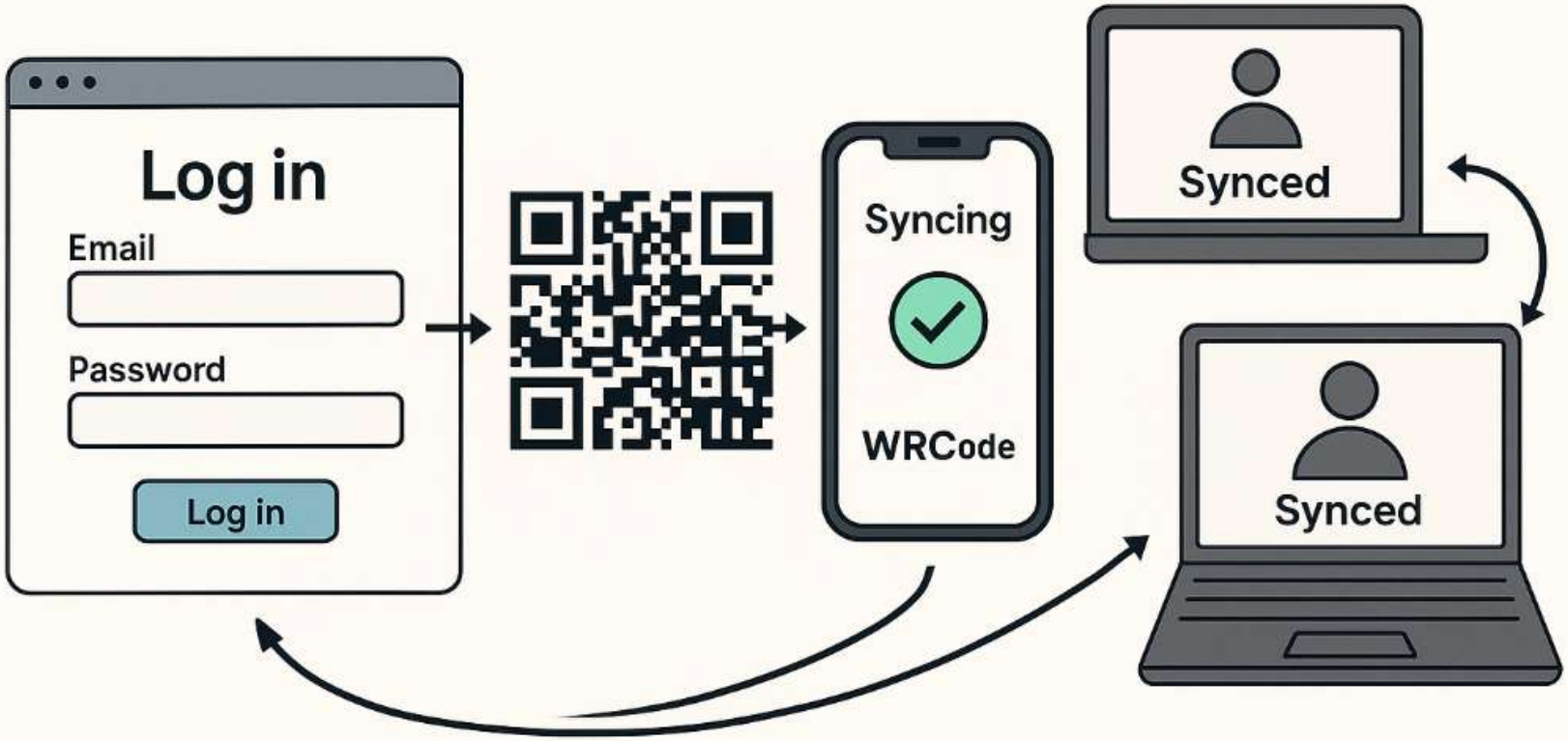
🌐 Open Source Deserves Real Security

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.

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., OpenGiraffe) 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 OpenGiraffe-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 OpenGiraffe 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.

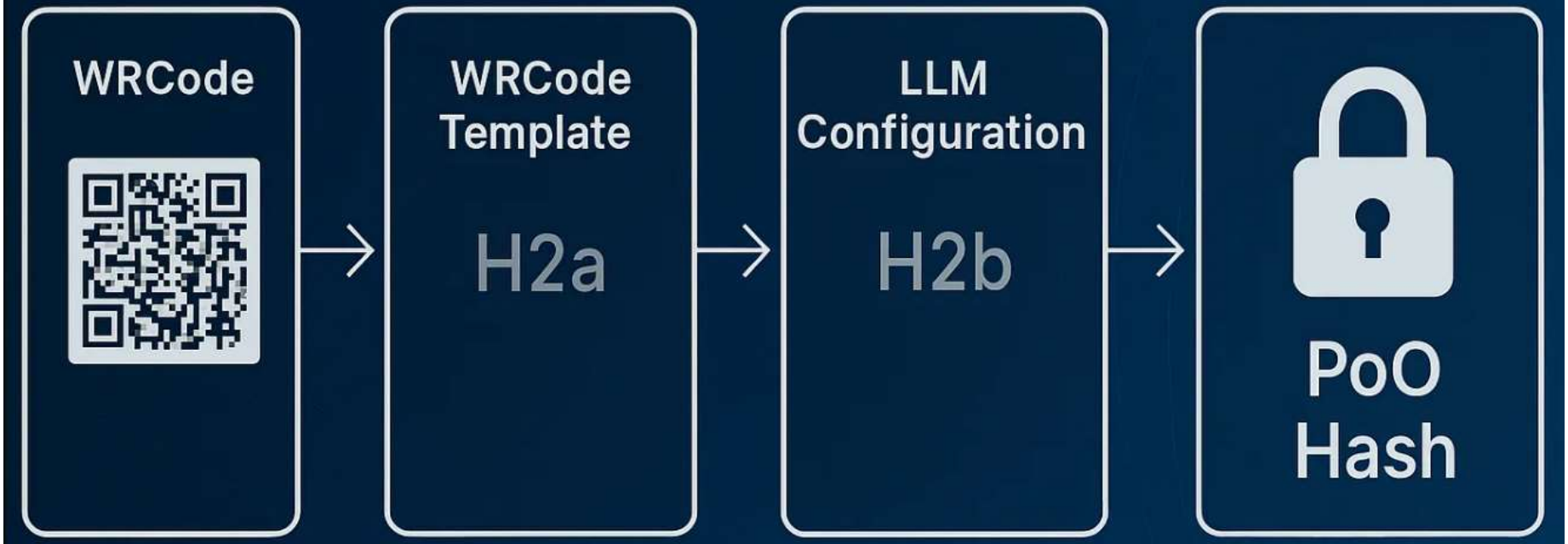
Once access is granted, a secure session token is issued. This token is cryptographically bound to the user's hardware, software instance, and verified WRCode.org account. It cannot be reused outside its original context, making it effectively useless if stolen. These tokens are fully revocable at any 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, OpenGiraffe 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-Orchestration

WRCode templates combined with AI and cryptographic verification



🔐 Proof of Orchestration (PoO) — 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 Orchestration (PoO) 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, PoO 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.

PoO 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 PoO, as local-only inference offers stronger and more verifiable protections. PoO provides a cryptographic receipt of the session—anchoring the declared input, reasoning context, and output trace in a tamper-proof, decentralized manner.

Importantly, PoO is **not a mechanism for deterministic output reproduction** , especially when using LLMs. Instead, it serves to prove:

- What WRCode was scanned
- Which templates 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

📦 PoO 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 PoO trace, and is confirmed using zero-knowledge proofs via cryptographic anchoring—ensuring that no PII is ever exposed during identity verification or orchestration logging

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 PoO.

We focus here on the practical integration of PoO into two core inventions: **WRCode-triggered workflows** and the **OpenGiraffe 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, PoO 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 PoO 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 PoO-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 PoO (Proof of Orchestration) that verifies the whole process is secure and verifiable

📜 What Is Proof of Orchestration (PoO)?

PoO 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 PoO
- That no part of the experience was altered or misused

Users can download their own `.poo.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

ℹ 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 PoO. 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 PoO 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 PoO 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 PoO 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 (PoO Evidence)

The recommended configuration is included as part of the PoO 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 guarantees that any execution claiming to represent a specific WRCode project follows the originally declared technical conditions, 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": [
  {
    "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": "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 wrcode.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 PoO. 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 Berin Central Station every 5 minutes.",

"output_hash": "bd83fe8a... (truncated)",

"linked_proof_of_orchestration": "abc123-poo"

}

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
    - PoO 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
    - PoO 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 templates 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 PoO 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 PoO 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
- PoO embeds these identities to enable full traceability and prevent impersonation or silent tampering.

📊 Practical Hashing and Trust Model in PoO

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

✳ What PoO Can and Cannot Guarantee

Aspect	Verified by PoO?	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 PoO scope

PoO 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, PoO 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 guarantee that these outputs:

- Followed a consistent set of logic or business rules;
- Used approved AI models or orchestration templates;
- 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 **OpenGiraffe** orchestrator—including workflow templates, user-defined settings, and AI model setup—can be cryptographically sealed using the PoO 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-Orchestration

A valid PoO consists of the following components:

📊 Structure of a Proof-of-Orchestration (PoO)

Component	Description	Hash ID / Trust Layer
Input Context Runtime Session	Original user input or task prompt	H1 = SHA256(input)

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 / PoO 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 PoO
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 = 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 **OpenGiraffe** orchestration system, **PoO** 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 OpenGiraffe with optional human interaction and AI agents. The result—an optimized suggestion, recommendation, or output—is then documented cryptographically using the PoO format.

A `.poo.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 **PoO**, 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 **PoO** 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 **OpenGiraffe** 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-Orchestration 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 PoO 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

... OpenGiraffe can offer **end-to-end traceability** — from user input, through AI optimization, to decentralized action. PoO adds an **execution-time integrity layer** to this setup, providing real-world verifiability for each decision.

7. Closing Note

Proof-of-Orchestration 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.

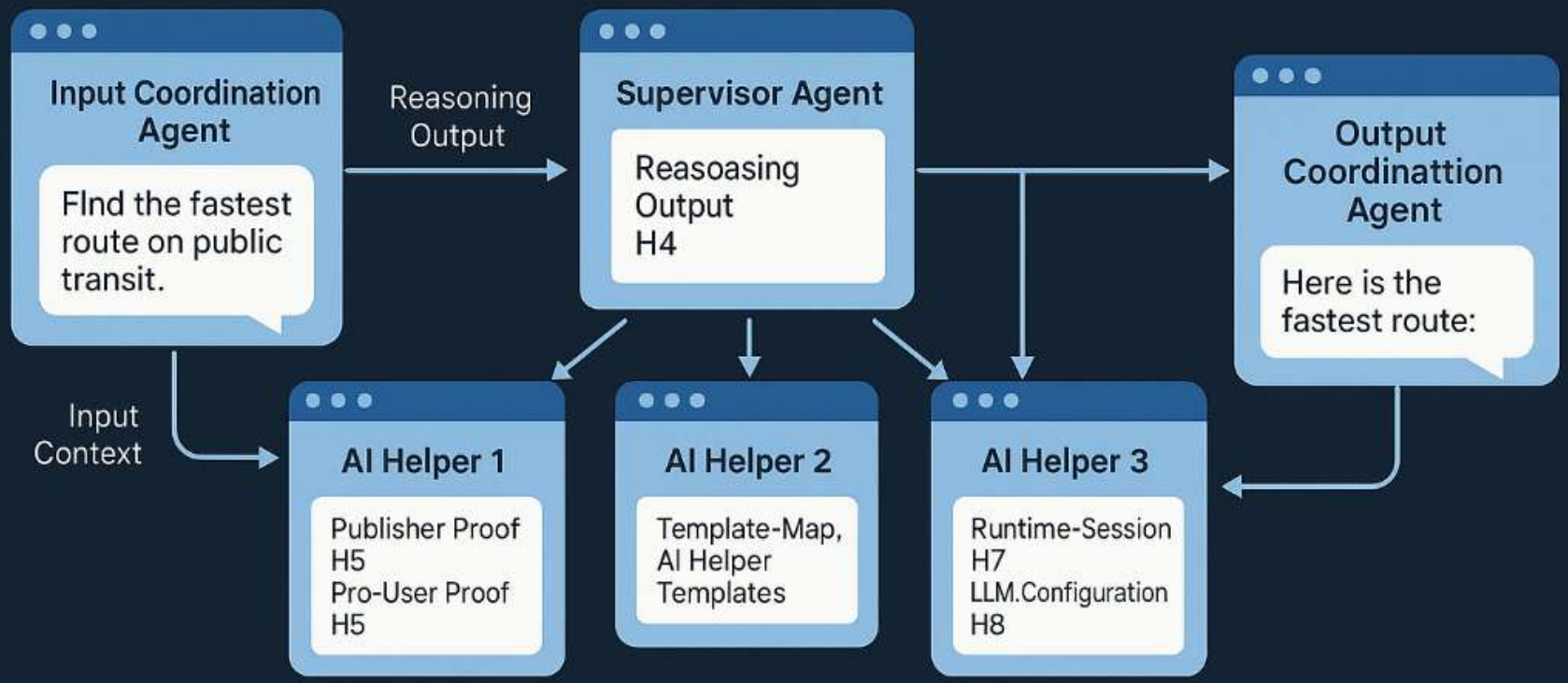
While further research is needed for formal ZKP extensions or tokenized enforcement, PoO provides a practical foundation — especially in WR Code-powered orchestration flows.

We do not claim novelty of the PoO mechanism in a legal or academic sense; instead, we present this as a **pragmatic approach** to enhancing trust and transparency within our open-source 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.

OpenGiraffe – Open-Source Browser-Based AI Agent Orchestration Framework

Adaptive Agent Architecture for Context-Aware Reasoning



Adaptive Agent Architecture for Context-Aware Reasoning

🤖 OpenGiraffe: Real-Time AI Orchestration for Mobile, AR, Robots, and Desktop—With Predictive, Privacy-Preserving Automation

OpenGiraffe 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 OpenGiraffe 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 OpenGiraffe 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: **OpenGiraffe** 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 OpenGiraffe 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 **OpenGiraffe** 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 **OpenGiraffe**—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 **OpenGiraffe** 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 **OpenGiraffe** 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 **OpenGiraffe** 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:

- **OpenGiraffe 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 in this stack can be self-hosted and are either open source or source-available:

- Optimando.ai (frontend orchestration layer),
- local LLM backends (e.g., via Ollama),
- vector databases (e.g., Qdrant, Postgres),
- and the automation layer (e.g., n8n).

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 — preserving transparency and enabling complete control over system behavior in professional environments.

✅ Summary

The **OpenGiraffe** 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 **OpenGiraffe** 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 open-source 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 open source, 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 tabs. 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)**

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, open-source 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.

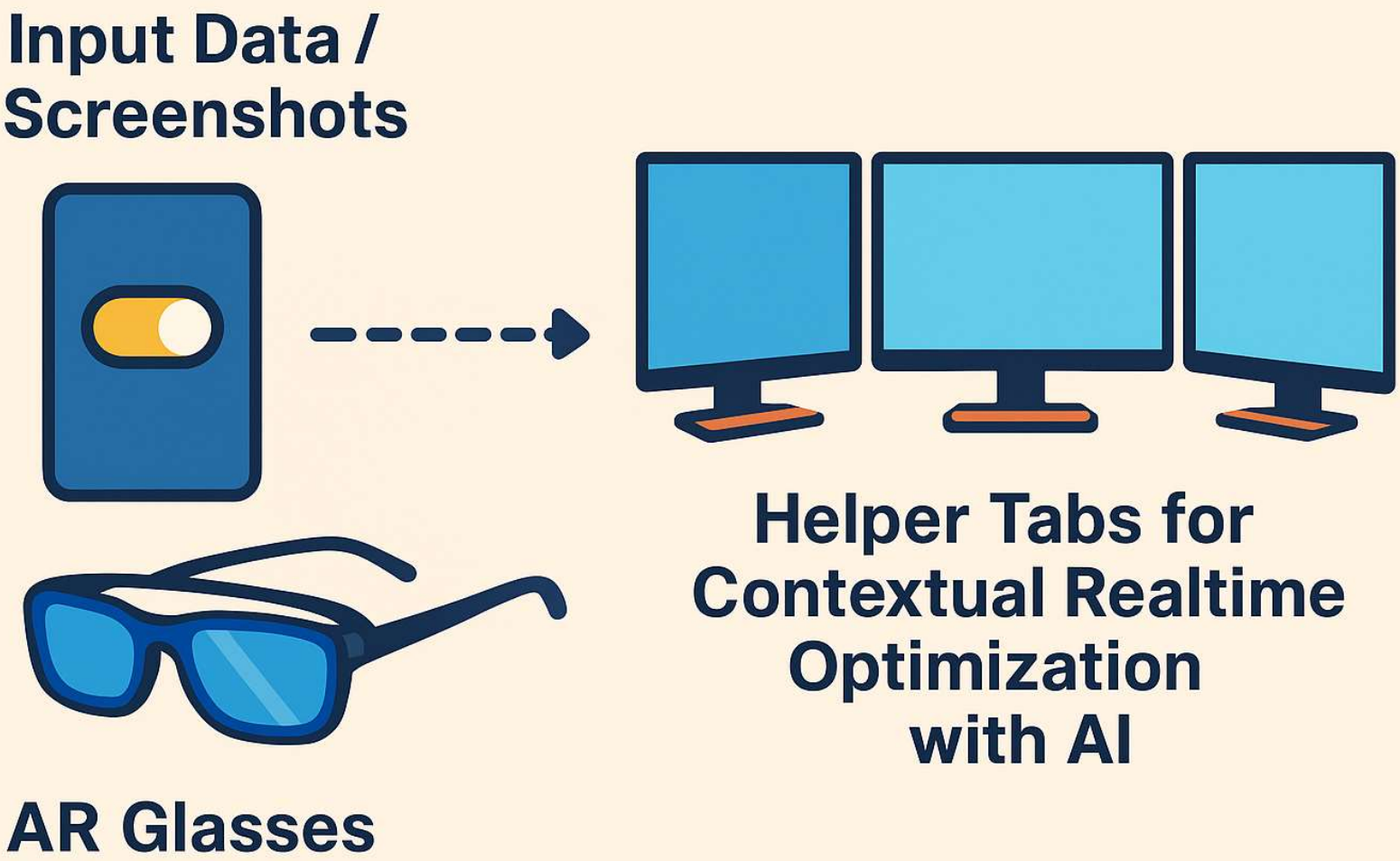
Users may operate directly at the orchestration workstation or remotely initiate tasks from other locations. The system supports both **autonomous execution** and **human-in-the-loop workflows**, enabling oversight, corrections, or adaptive feedback. This flexibility allows real-time augmentation, background 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
Helper Tabs for
Contextual Realtime
Optimization
with AI
AR Glasses

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 Optimando.ai server**. The coordination logic and agent communication infrastructure are open-source 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

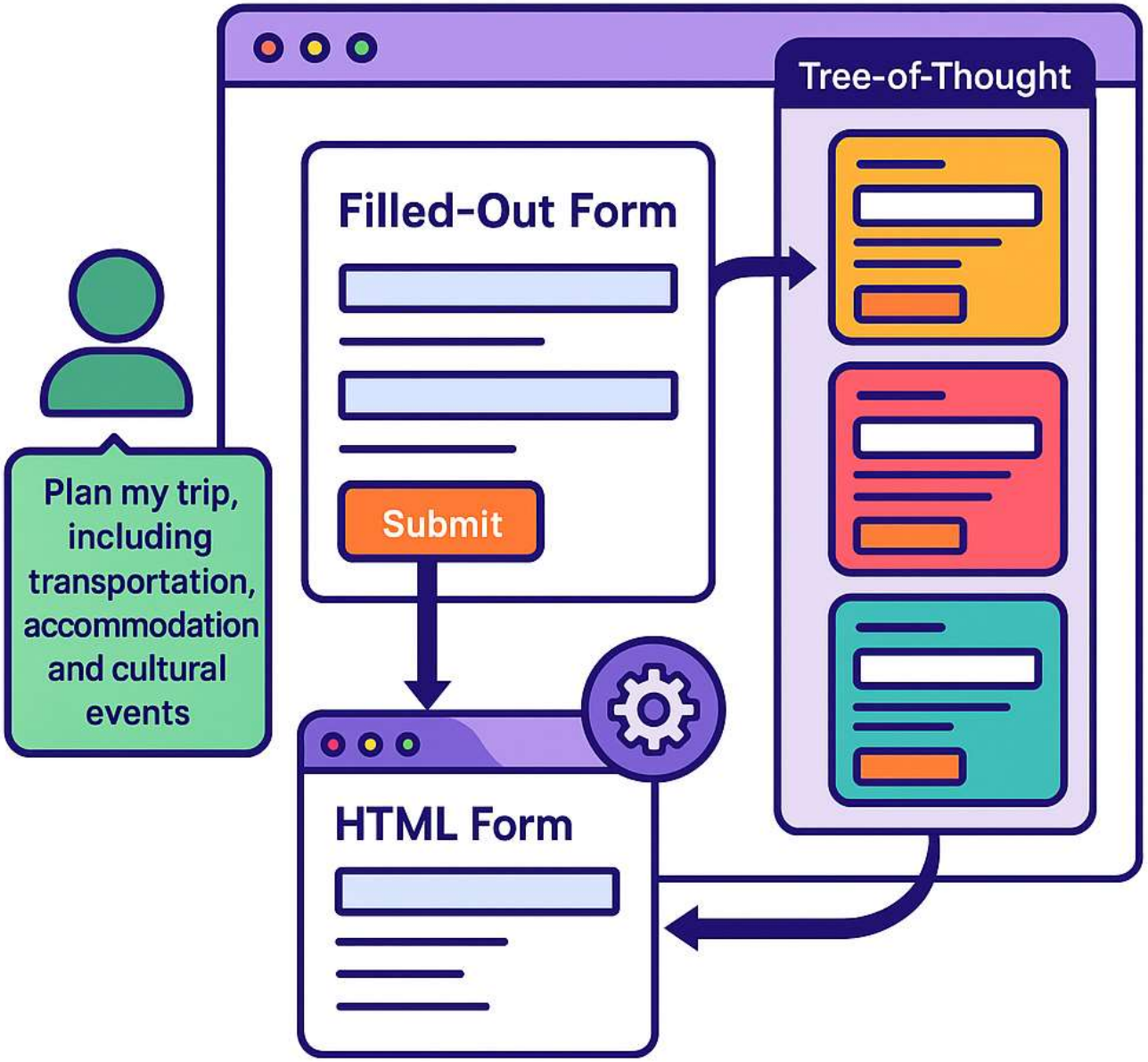
does not operate or provide any backend services that receive, log, or process this data. The system's observation is also strictly limited to in-browser content in explicitly configured tabs. There is **no tracking of desktop activity or full-device behavior**.

Each helper tab in the browser may be assigned a task-specific AI agent, 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.


Tree-of-Thought
Filled-Out Form
Submit
Plan my trip, including transportation, accommodation and cultural events
HTML Form

🔧 DOM Execution in Background Helper Tabs

Each helper tab 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 **OpenGiraffe** 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 OpenGiraffe'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, **OpenGiraffe'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, **OpenGiraffe** 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—**OpenGiraffe** could enable seamless, privacy-respecting automation that bridges physical and virtual environments. These ideas illustrate potential future extensions beyond **OpenGiraffe'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 an open-source 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
Master Tab	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. Because the Master Tab handles real-time input directly from the user and external sources, *optimando.ai* 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. *optimando.ai* provides tooling to structure, automate, and augment downstream tasks — but **cannot enforce privacy at the source input level. The user remains in control.*
Helper Tabs	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.

Component	Privacy Role
	However, the degree of masking depends on user configuration and workflow context. *optimando.ai* 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.

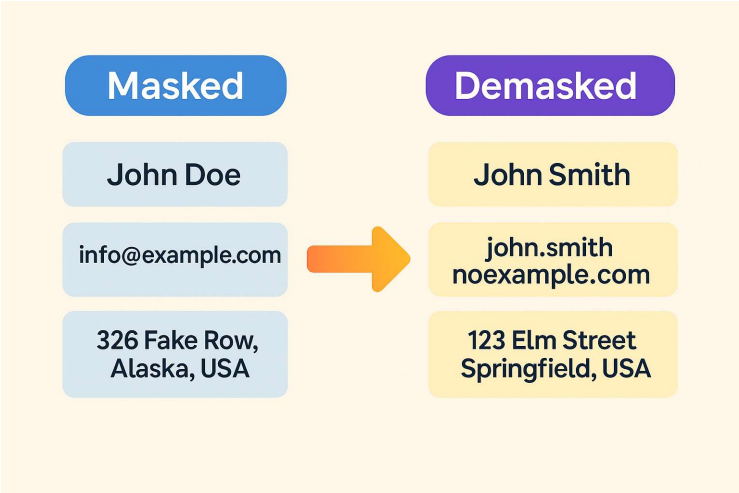
This orchestration strategy allows privacy protection at the execution level, not just via policy.

🛡 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 an open-source, 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 open-source 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 OpenGiraffe 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 OpenGiraffe'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 OpenGiraffe, 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

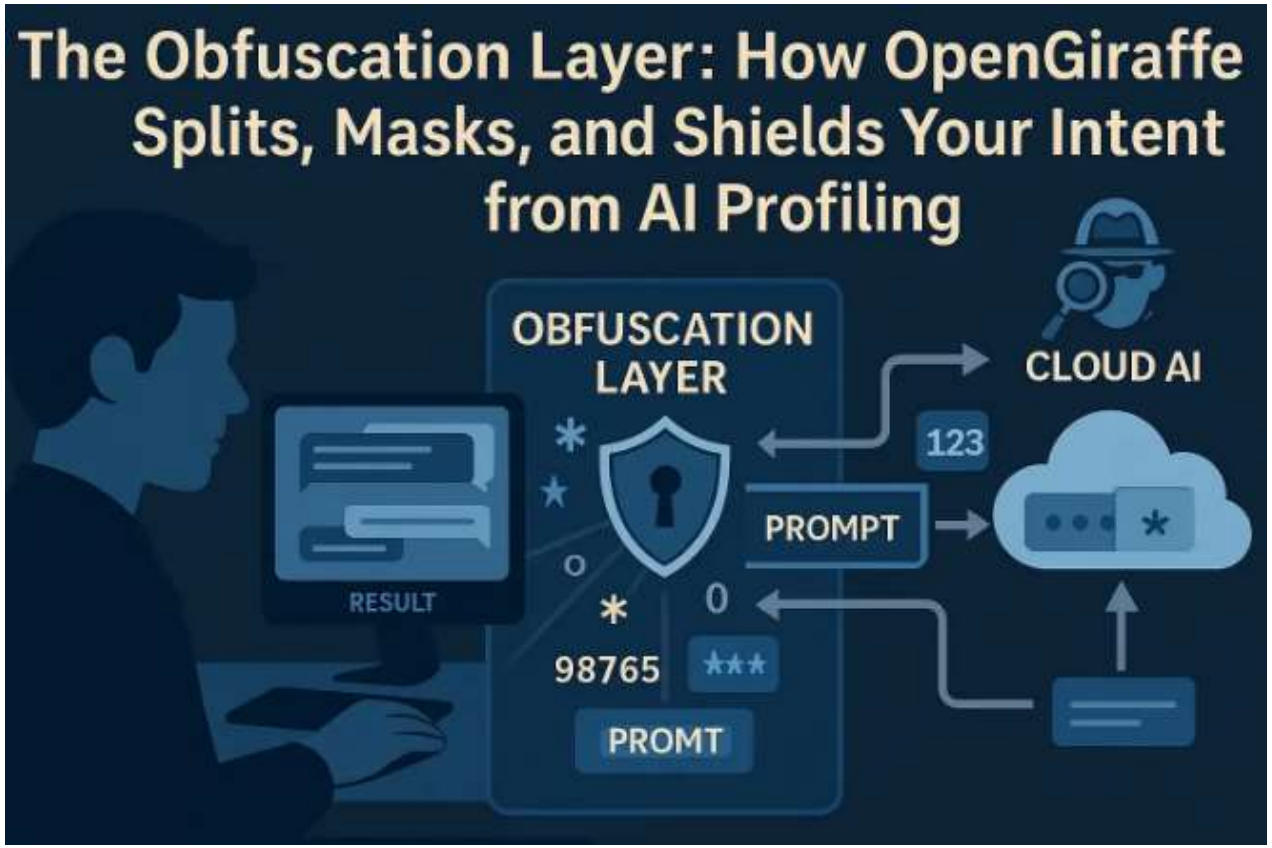
The Obfuscation Layer is a live research field. It continues to evolve with:

- **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

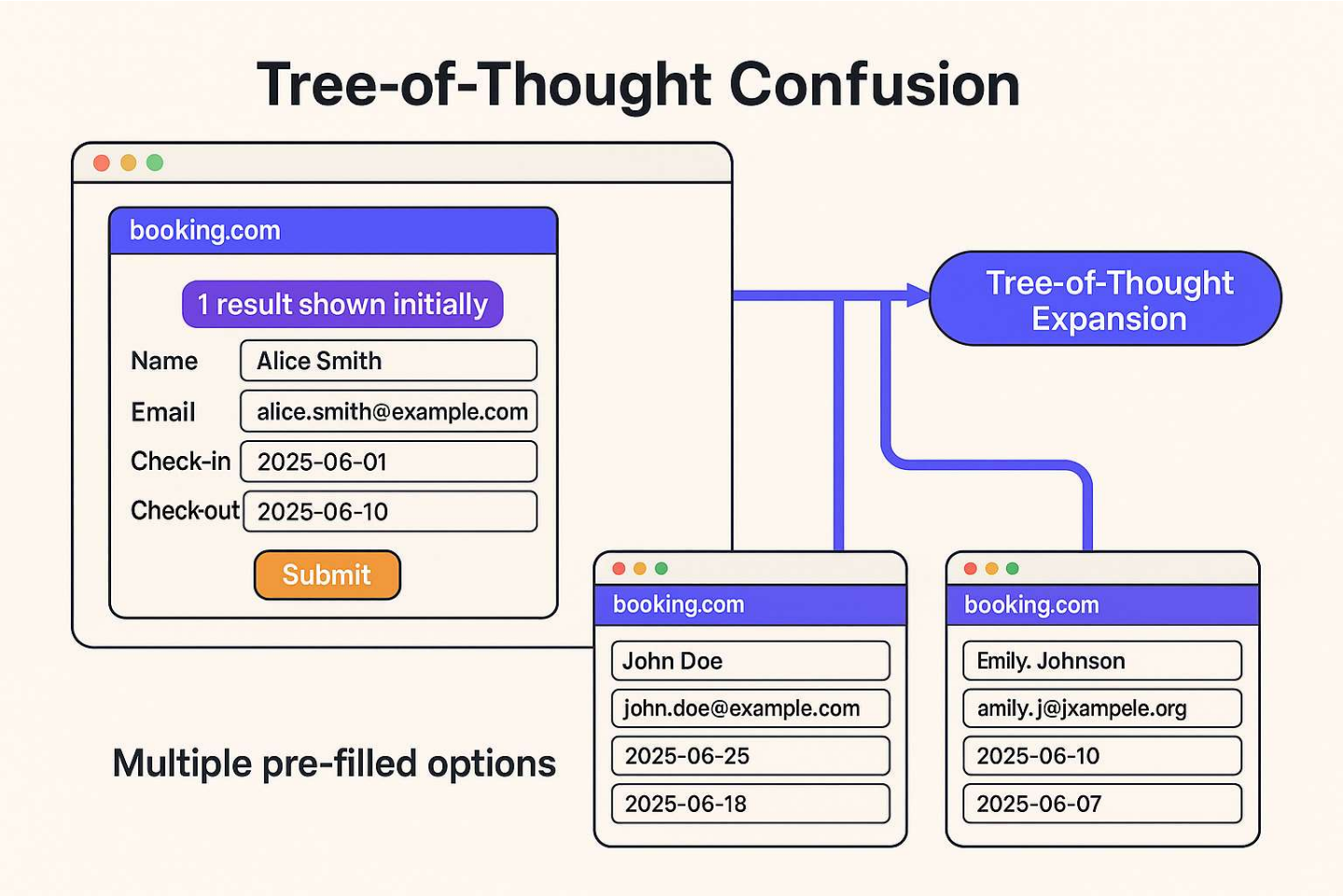
Thanks to its distributed helper-tab design, *optimando.ai* enables a novel privacy feature: intent obfuscation through variant generation.

- **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. Open Source, always ready to improve and evolve.



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.

Conceptual Feature: On-Premise Augmentation Overlay for Masked Reasoning

As part of future development, we propose an experimental feature designed to improve the interpretability and safety of decision-making under data masking: a Master UI overlay that dynamically highlights critical input fields and allows users to simulate randomized variations of masked data locally.

This concept—referred to internally as the Augmentation Overlay—would provide a user-facing mechanism to explore how masked or obfuscated data influences the system's reasoning path. The feature is intended only for on-premise execution and is designed to ensure that no sensitive data is ever transmitted or logged externally. The overlay itself could be toggled on or off. After all we aim for a fully autonomously real-time user intent and prediction loop with forward-thinking real-time support.

Key Ideas Behind the Overlay Concept

- **Dynamic Field Highlighting**
  The Master UI would automatically detect which input fields are involved in the current masking/demasking logic and visually augment them using a transparent overlay. Fields may be color-coded or animated to indicate their importance in the current reasoning context.

- **Simulated Input Randomization**
  Users can press a button to generate plausible random values for any field involved in the masking logic. This enables counterfactual simulations: "What would the system conclude if these values were different?"

- **Human-in-the-Loop Reasoning Validation**
  By observing how simulated changes affect the output, users can monitor whether the reasoning engine reacts in unexpected or overly sensitive ways. This strengthens transparency and helps identify fragile dependencies in automated decision logic.

- **Field Sensitivity Feedback (optional extension)**
  Future iterations of the concept could introduce a sensitivity score per field, highlighting which data points have the greatest impact on logic flow when masked or altered.

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 orchestration system's source code is published under the **GNU Affero General Public License v3 (AGPLv3)**, reflecting a strong commitment to **open-source principles** and ensuring derivative works remain open, particularly in network environments. The conceptual framework and accompanying documentation described in this paper are licensed under the **Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0)**. This dual licensing model for the software, along with the **CC BY-SA 4.0** for conceptual works, ensures both robust open-source integrity and safeguards for attribution and openness across all project artifacts. For commercial, closed-source, or enterprise environments requiring alternative terms (including specific UI attribution mandates), a separate commercial license is available.

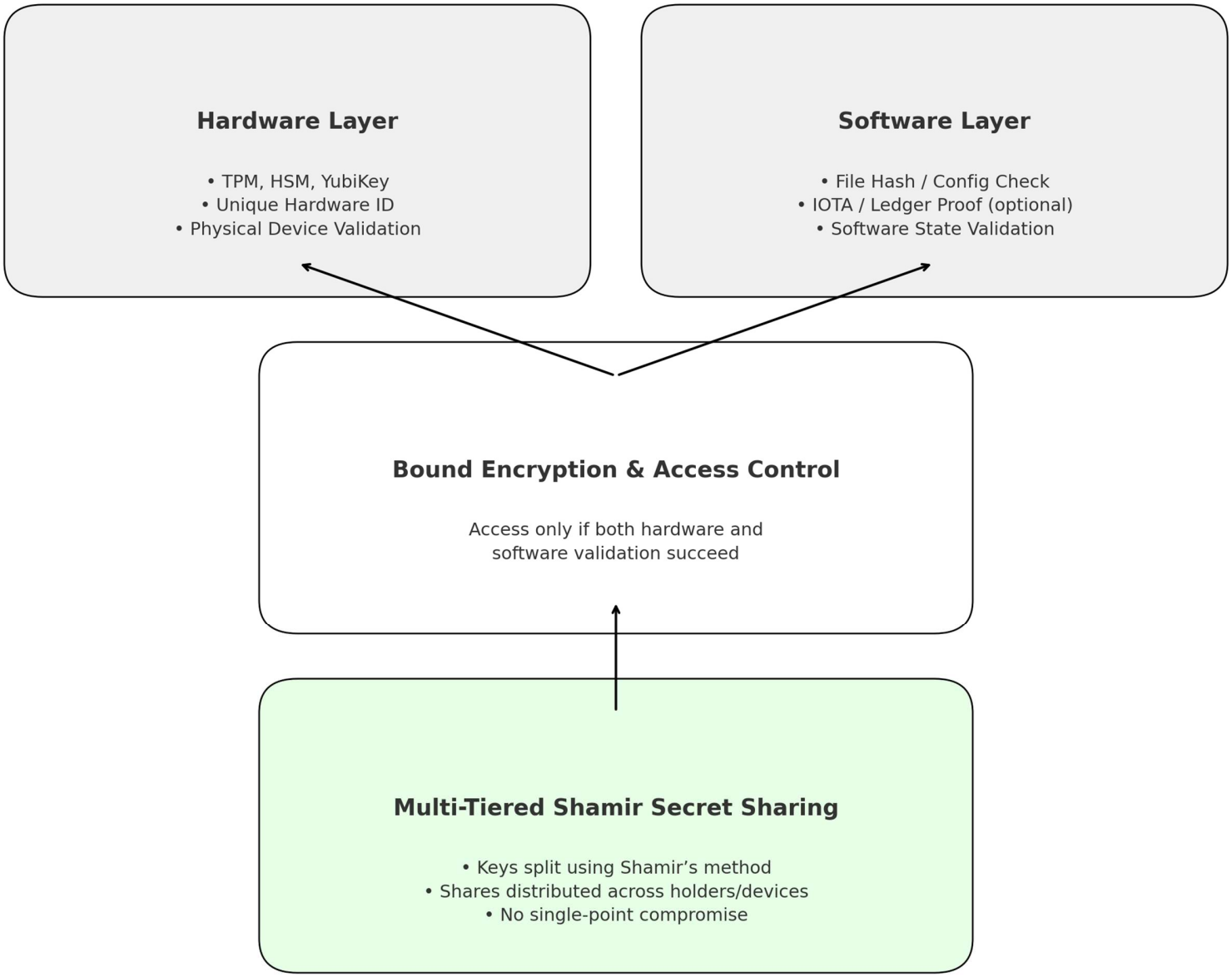
OpenGiraffe: A Comprehensive, Future-Proof Security Framework for Local AI Orchestration in High-Risk Environments

Abstract: **OpenGiraffe** is an open-source, 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. **OpenGiraffe** 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 (blockchain, DLT) ensure integrity in connected systems, while air-gapped environments achieve similar guarantees through local cryptography and secure hardware (e.g., HSM, TPM).**
- **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:
OpenGiraffe 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):
OpenGiraffe 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, **OpenGiraffe** 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:
OpenGiraffe 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. Optional Ledger Anchoring, Template Authenticity & Zero-Knowledge Proofs via Solana/IOTA

OpenGiraffe 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 OpenGiraffe project is committed to ongoing research into quantum-proof cryptography, Zero-Knowledge Proofs, and AI governance to ensure long-term security and innovation leadership.

OpenGiraffe 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)

OpenGiraffe 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 **OpenGiraffe** 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, **OpenGiraffe** 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 OpenGiraffe is freely deployable as open source, 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:
OpenGiraffe 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.

OpenGiraffe and its tools are open-source and trustless—enabling everything from easy setups to tamper-proof enterprise solutions. Future-ready LLMs without blind trust.

LetMeGiraffeThatForYou: Visual Question Capture and Automated AI Orchestration Trigger in OpenGiraffe

As part of the **OpenGiraffe** open-source 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 **OpenGiraffe**-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 **OpenGiraffe'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:**

- **OpenGiraffe** 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 **OpenGiraffe**: 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 OpenGiraffe 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 **OpenGiraffe** orchestration framework. The underlying concept, technical design, and process constitute original intellectual property (IP) within the **OpenGiraffe** 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.

Decentralization Roadmap for OpenGiraffe's WRCode Architecture

Introduction The OpenGiraffe 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, OpenGiraffe 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 and manifest hosting via WRCode.org (centralized starting point)
- Publisher verification and orchestration hosted on optimando.ai 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)*

- Open-source 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.

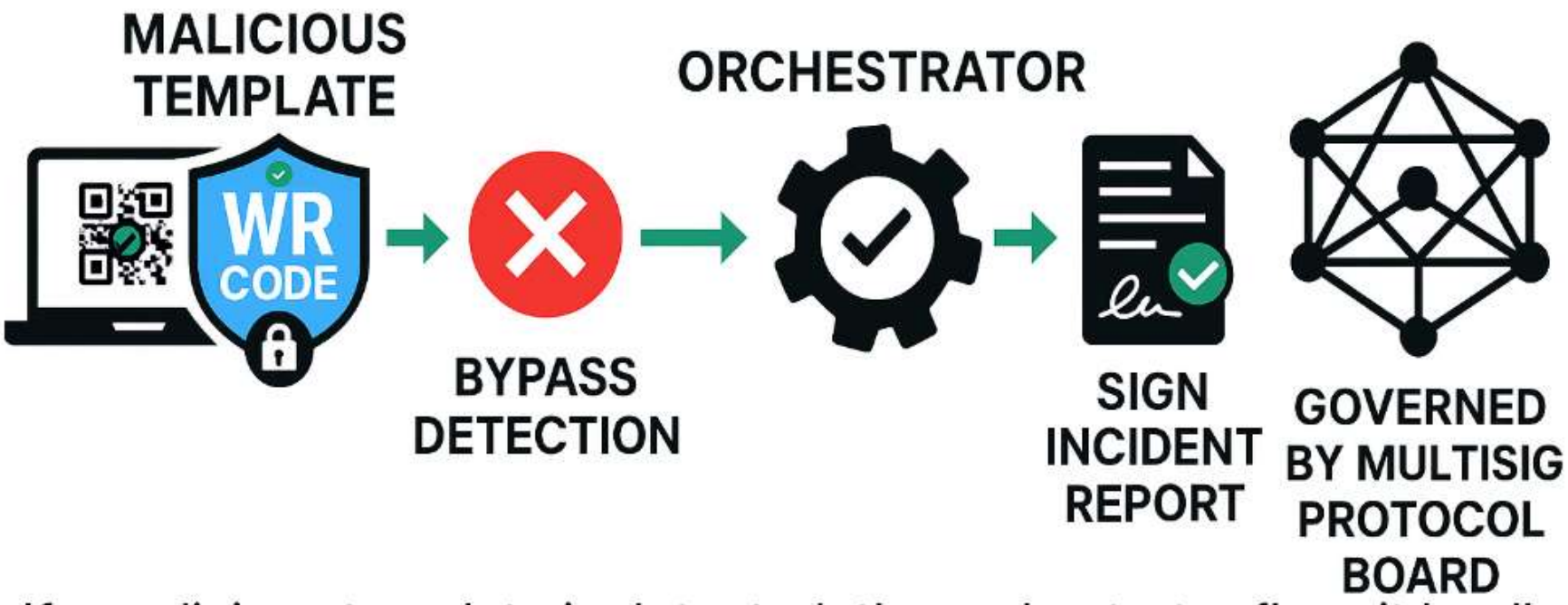
At this stage, WRCode would evolve into a neutral public protocol. The creator would no longer be a bottleneck or a target — but may remain a critical innovator and economic operator.

Conclusion: Long-Term Trust Through Transparent Design

The future of WRCode and the OpenGiraffe 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.

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

⚖️ Strategic Trade-Offs and Professional Target Group

Optimando AI's OpenGiraffe 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 ***OpenGiraffe*** – 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, open-source, 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.compatents.google.com. 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.compatents.google.com. 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.compatents.google.com. 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.compatents.google.com. 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.compatents.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.compatents.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.comstrategemist.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.comstrategemist.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 systemsmdpi.commdpi.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 privacymdpi.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.commdpi.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 ledgerproveai.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 toolsproveai.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 dataarxiv.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 *1000× overhead* in time and memoryarxiv.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 resultsarxiv.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 resultarxiv.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 proofsethresear.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" securityarxiv.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 watchersmedium.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 resultarxiv.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 chainmedium.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 resultsmedium.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 entryfeng.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 ledgerfeng.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 producefeng.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 systemnews.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 cloudmedium.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-shirtmedium.datadriveninvestor.commedium.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 usermedium.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 ideasmedium.datadriveninvestor.commedium.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 phoneifttt.comifttt.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 ownershippatents.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 statechain.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 consensusmedium.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 latermedium.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.compatents.google.com.
2. US Patent 11,494,171: *Decentralized platform for deploying AI models* (2022) – using NFTs and blockchain for AI model validationpatents.google.compatents.google.com.
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.
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.
7. Conway, K. *et al.* (2024). "opML: Optimistic Machine Learning on Blockchain" (arXiv 2401.17555)arxiv.orgarxiv.org; 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.
9. Morpheus.Network (2019). *Supply Chain Platform Launch* – QR code triggers and blockchain footprints in workflowsnews.morpheus.network.

10. **Angeri, H. (2023). "Every Object is Intelligent – Enabling All Objects with AI Agents" (Medium)medium.datadriveninvestor.commedium.datadriveninvestor.com.**

11. **Feng, X. (2018). "Data Integrity and Data Lineage by using IOTA" – blog demonstrating IOTA's use in IoT data verificationfeng.lu.**

Google DeepMind's Project Marinertechcrunch.comtechcrunch.com, OpenAI's Operatortechcrunch.comtechcrunch.com, UCSD's Orca browser researcharxiv.orgarxiv.org, the Nanobrowser open-source projectgithub.comgithub.com, Opera's AI initiatives (Aria and Neon)press.opera.compress.opera.com, and curated lists of web AI agents and automation toolsgithub.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

An **open-source, 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 open-source 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
Open-source	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 an **open-source, 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.**

Tree of Thought
Analytics
46%
87.8%
5%
6%
AI Orchestration
Agants
Task Flow

🧠 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.

🛠 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 **OpenGiraffe 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)

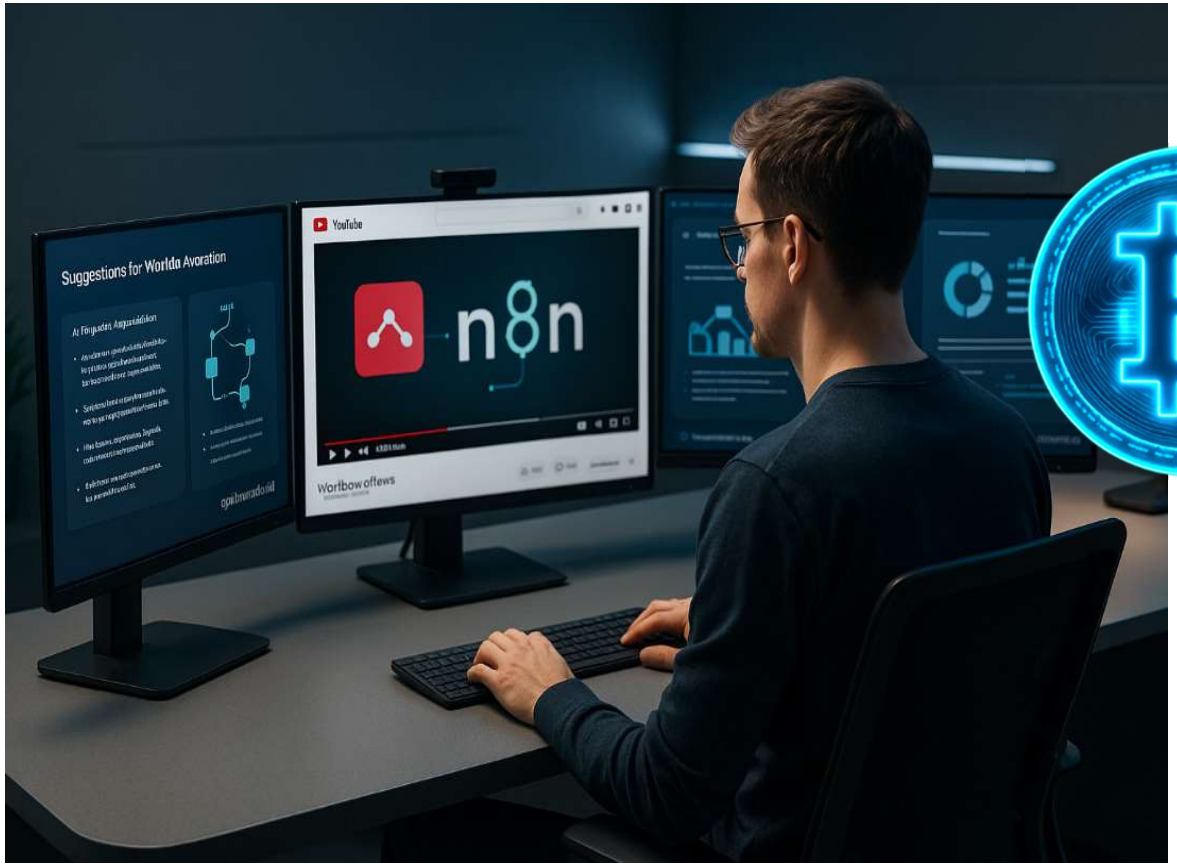
- **Published on Zenodo and GitHub**

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.

🏁 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 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 open-source 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.

Concept Timestamped on Bitcoin



© 2025 Oscar Schreyer / O.S. CyberEarth UG (haftungsbeschränkt) — published as part of the Optimando.ai project.

This work documents the OpenGiraffe© architecture, including QRGiraffe© — an open-source 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.

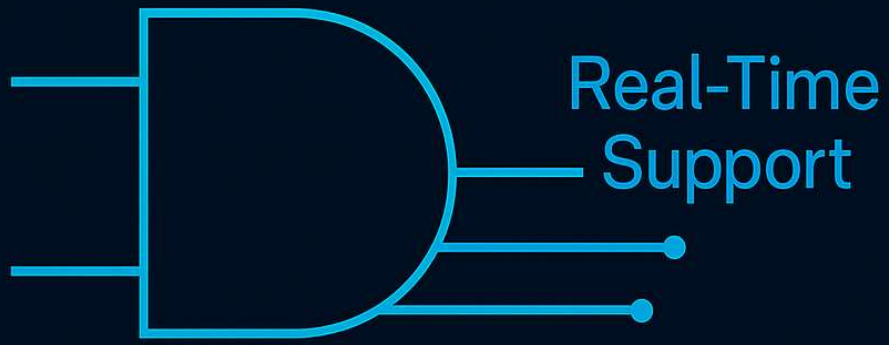
Certain components — such as source code, automation templates, or diagrams — may be subject to additional licensing terms, including the GNU Affero General Public License (AGPL) v3.0. Please refer to the included license files for specific details.

Proof of authorship has been cryptographically timestamped using OpenTimestamps.

License references:

- Creative Commons (CC BY-SA 4.0): https://creativecommons.org/licenses/by-sa/4.0/
- GNU AGPL v3.0: https://www.gnu.org/licenses/agpl-3.0.html

OPTIMANDO AI



© 2025 Optimando AI