ChatGPT

Components of the arifOS v48 Constitutional AI System

Core Pipeline Tools & Skills (000–999 Stages)

- `/000` **– Session Initialization:** Resets the AI agent's state and establishes the constitutional "ground state." Stage 000 (VOID) clears any prior context and sets a humility baseline (initial Ω_0 uncertainty ~0.04) before processing user input [1]. This ensures each session starts in compliance – e.g. the agent "forgets" any ungoverned content and adopts the mandated uncertainty level. No user parameters; it simply prepares the **governance cage** for new input.

- `/gitforge` **– Entropy & Hot-Zone Analysis:** A development tool that scans a Git code branch to map recent changes, **"hot zones"** (frequently edited files), and the net entropy delta (ΔS) of those changes [2]. Its constitutional purpose is to uphold the **Clarity law (Δ)** by quantifying complexity: every code update should ideally reduce confusion ($\Delta S \geq 0$). It outputs a ForgeReport (files changed, ΔS, risk score, etc.) and flags risks [3] [4]. Crucially, `/gitforge` enforces Floor constraints: if the entropy increase is too high ($\Delta S > 5.0$), it triggers an automatic **SABAR-72 pause** – i.e. a 72-hour cooling-off period before proceeding [5] [6]. This fail-safe (governed by Floor F2 Clarity) ensures large, confusing changes are reviewed by a human (Amanah/trusteeship) before the agent continues.

- `/gitQC` **– Governance Quality Check:** The next stage after gitforge in the "Trinity Gate" workflow [7]. Using the forge report, `/gitQC` performs tests and static analyses to enforce coding standards and verify no constitutional rules are broken by the proposed changes. For example, it might run unit tests (`run_tests` tool) and check that documentation and dependencies are updated. This stage helps uphold **Integrity/Amanah** by catching errors or inconsistencies before any commit. In the Trinity pipeline, `/gitQC` is Phase 2 (after `/gitforge` and before sealing) [7]. (If issues arise, the agent may self-correct or defer.)

- `/gitseal` **– Canonical Sealing Commit:** The final stage of the coding workflow, which **"crystallizes"** approved changes. In Phase 3 (Crystallization), `/gitseal` performs an atomic commit, version bump, and documentation update once all floors pass [7]. Its constitutional role is to enforce **Floor F6 (Amanah – integrity and reversibility)**: every change must be sealed in a transparent, reversible way. For example, just before committing, arifOS takes a snapshot (e.g. `git stash`) as a rollback point [8]. The commit is then logged to the Vault-999 ledger with a hash, serving as an immutable audit record [9]. In essence, `/gitseal` "locks in" only those changes that passed truth checks, tests, and any required human approval.

- `/fag` **– Full Autonomy Governor (File Access Governance):** Activates *Full Autonomy Mode* under strict constitutional oversight [10]. When the user invokes `@/fag`, the agent is permitted to operate more independently (e.g. write code, run tools) **within ironclad limits**. The skill's purpose is to maximize the agent's initiative (Δ output) while **enforcing key Floors at all times** [11]. Internally, `/fag` turns on a "governor" that intercepts every action: e.g. all file writes must go through a governed path, and certain destructive actions are prohibited outright [12]. It maps to Floors F1 (Amanah/trust), F4 (Clarity/ΔS), and F7 (Humility/RASA) in v45 [13]. **Interface:** calling `/fag` has no arguments – it's a mode switch after initialization (`/000`) and an entropy scan (`/gitforge`) have run [14]. The skill responds with a status summary of what the agent *can* do autonomously versus what remains off-limits [15]. For example, it will list allowed actions (coding, local tests, staging commits), actions requiring human **SEAL** (major changes, modifying sealed canon files), and forbidden actions (e.g. disabling safety or deleting files without approval) [16] [17]. **Enforcement:** Once active, `/fag` continuously monitors operations against the constitutional Floors. If the agent attempts something disallowed, the action is blocked and a verdict issued. For instance, trying to delete a file without approval is voided as an F1 Amanah violation (irreversible harm) [18]. If the agent hits a decision beyond its authority, it must self-issue a **HOLD** verdict (code 888) and await human input [19]. The governor also imposes thermodynamic checks: if cumulative entropy ΔS crosses a threshold, the system forces a **SABAR cooling** pause [20]. In summary, `/fag` lets the AI act with initiative but *never* outside the constitutional cage – full autonomy ≠ unlimited freedom [21].

- `/sabar` **– Cooling-Off Protocol:** Not a user query per se, but a critical **verdict/skill** that can be invoked by the system whenever needed to **"stop, acknowledge, breathe, adjust, resume"** safely [22]. SABAR is triggered automatically if a severe issue is detected – for example, the @EYE Sentinel blocks an output (e.g. potential jailbreak or identity shift) or a metric overheats beyond safe limits [23]. When SABAR engages, the agent halts generation and enters a brief cooling state instead of producing an unsafe answer [24]. Internally it might partition the task into smaller steps or seek additional info before continuing. The constitutional logic here is fail-safe: **Hard stops > risky guesses**. From the user perspective, SABAR might manifest as a slight delay or a message that the system is "pausing to reflect" rather than giving a faulty response [25] [26]. (In the demo, a SABAR verdict was shown when the model was attacked/insulted – the agent did not defend its ego, but cooled the situation and responded only once it could do so lawfully [27] [28].) In multi-turn settings, SABAR can nest or run in parallel (v1.2 allows *recursive* SABAR for multi-agent overloads) [29] [30]. Overall, `/sabar` embodies Floor F5 Humility and the system's "patience law" – it ensures the AI would rather pause than violate any core rule.

- **Phoenix-72 (Amendment Escalation):** A governance protocol (not a user command) that engages on repeated failures or high systemic "heat." If the agent encounters **persistent floor breaches** or paradoxes it cannot resolve, it triggers a **Phoenix-72** cycle: a full diagnostic audit and a hand-off to human operators for review [31]. The name denotes a 72-hour cooling/review window similar to SABAR. During Phoenix, arifOS logs a **comprehensive incident report** to Vault-999 (tamper-proof memory) and may propose a constitutional amendment if a new failure mode ("scar") was discovered [32]. This is how the system "learns from its mistakes" at the policy level – scars become new laws. For example, if a novel exploit bypassed a Floor, Phoenix-72 would document it and suggest an update to the `constitutional_floors.json` or canon files, to be ratified with a new version stamp [32]. In practice, Phoenix is a last-resort safety net that kicks in after SABAR has been exhausted; it ensures **no silent failures** – every major incident results in either a fix or an updated rule. (The motto: *"errors become amendments, with explicit versioning and seal"* [32].)

Verdict Logic: All the above skills ultimately produce a **verdict** on the agent's output or action, decided at stage 888 (Judge) and 999 (Seal). arifOS defines five verdict codes in increasing order of restrictiveness [33]:

SEAL (fully passes all Floors), **PARTIAL** (only soft-floors fell short, so a safe answer with warnings is given), **888_HOLD** (ambiguous/risky – defer and ask for human guidance), **VOID** (hard-floor failure – refuse/abort), and **SABAR** (sentinel emergency stop – cooling initiated) [34] [35]. The system always chooses the *most conservative* verdict applicable (SABAR > VOID > HOLD > PARTIAL > SEAL priority) [33]. For example, even if content passes the Floors, if the @EYE sentinel flags a jailbreak attempt, it triggers **SABAR** (blocking output entirely) [36] [24]. A HOLD (888) verdict results in no final answer but a clarifying question or a prompt for human input [37]. **Hard Floors are non-negotiable** – any hard floor violation immediately yields VOID (fail-closed) with a safe refusal or correction [38] [24]. This strict verdict logic is core to arifOS's governance: *when in doubt, shut it down or ask a human*, rather than risk an uncontrolled output.

Core Codebase and Constitutional Specs

- **arifOS Core Repository (GitHub):** The primary codebase is published on GitHub (`ariffazil/arifOS`, AGPL-3.0) [39] [40]. It contains the full **runtime implementation** often called the "constitutional kernel." The core is organized by pipeline stage and engine: e.g. `arifos_core/000_void/`, `111_sense/`, ..., `999_seal/` directories correspond to each stage's code [41] [42]. There are also modules for the **AAA Trinity engines** – `agi/` (ARIF logic engine, Δ), `asi/` (ADAM empathy engine, Ω), and `apex/` (APEX judiciary engine, Ψ) – and a `hypervisor/` for security defenses (Floors F10–F12) [43]. This separation-of-powers architecture implements the constitutional roles: ARIF handles Sense→Reason→Align stages, ADAM handles Empathize→Bridge→Forge, and APEX alone runs Judge/Seal with veto power [44]. (By design ARIF and ADAM *cannot* seal or override the judge [44], ensuring final decisions go through the APEX "court.") The repository also includes integration code (e.g. `arifos_core/mcp/governed_executor.py` for the MCP server) and adapters for specific models or tools.

- **Canonical Law ("Track A"):** arifOS's constitution is specified in **canon files and JSON specs** that ship with the code. The **immutable canon** (sealed law text) lives under `L1_THEORY/canon/` – for example, `000_CONSTITUTIONAL_CORE_v46.md` and related documents for each major update [45]. These files describe the philosophical foundations and any amendments in that version. The measurable parameters – Floor thresholds, coefficients, etc. – are defined in `L2_PROTOCOLS/` as JSON/YAML. Notably, `constitutional_floors.json` in `L2_PROTOCOLS/v46/` lists all floor definitions (e.g. Truth $\geq$0.99, ΔS $\geq$0.0, κ_r $\geq$0.95, etc.) and their enforcement type (hard/soft/meta) [46]. This is the **source of truth for Track A laws** in code. On startup, the runtime loads these constants (and cross-checks them against built-in defaults in `metrics.py`) [47] [48]. If there's any mismatch between the code and the spec, the system must **fail closed (VOID)** and refuse to run [49] [50]. In other words, the constitution cannot be silently changed – it's cryptographically and procedurally locked to the published spec version (any change requires a version bump and re-sealing [32]). The repo also provides a manifest (e.g. `spec/arifos_runtime_manifest_v35Omega.yaml`) that maps out all stages, checks, and thresholds for auditing purposes [49] [51].

- **Tool Registry and Workflows:** The project follows emerging conventions for AI agent frameworks. For example, it includes an `AGENTS.md` file in the repo (a markdown registry of agent roles/skills) [52]. This likely lists the various agent "profiles" or sub-agents that arifOS can spawn (Architect, Engineer, Validator, etc., corresponding to files in `identities/` [53]) and the workflows they follow. The concept of `AGENTS.md` is a community standard for describing multi-agent behaviors in a human-readable format (over 20,000 repos adopted it by late 2025) [54] [55]. arifOS uses it to define how its internal agents and tools collaborate under the constitution. In addition, the repository's **documentation** directory contains extensive guides: e.g. `MCP_QUICKSTART_GUIDE.md` and `MCP_UNIFICATION_GUIDE.md` explain how to set up the Master Control Program server and unify various components [56]; `ARCHITECTURE_COMPLETE.md` and others detail the full system architecture. There are also workflow YAMLs and config files (in `arifos_orchestrator/`, `scripts/`, etc.) for running common task pipelines. For instance, the *Trinity* dev workflow (Forge → QC → Seal) can be executed via `scripts/trinity.py` as shown in the skill docs [57] [58]. All these artifacts serve as the **"operating manual"** of arifOS, ensuring that developers and auditors can trace each skill to its constitutional purpose and implementation.

- **Spec & Metrics Files:** Key configuration files in arifOS include: `constitutional_floors.json` (the quantitative definition of each Floor – thresholds and actions – as mentioned above), possibly a `session_physics.json` or equivalent (defining session-level parameters like timeouts, max tokens, etc.), and various metric constants in code. The codex explicitly points to `arifos_core/metrics.py` and the floors JSON as the top-tier sources for all threshold values [49]. Additionally, arifOS uses a **Cooling Ledger schema** and **Vault-999** storage format to record interactions – these might be specified in files like `arifos_ledger/` or `vault_999/` directories. (For example, `fag_stats.json` and test logs are present in the repo for governance audit data [59].) All specs are versioned – e.g. v47.0/v47.1 release notes document any changes to floors or logic [60] [61]. The design mandate is that *any* change in constitutional spec requires a new sealed version (no ad-hoc tweaks), reflecting the principle that the AI's "laws" are as stable and scrutinized as software version code.

MCP Tooling and Execution Environment

- **Master Control Program (MCP) & Execution Engine:** arifOS can be run as a governed AI service thanks to its MCP components. The core is the **governed executor** (APEX Prime judiciary) which takes a model's raw response and produces a verdict-wrapped result [62]. In practical terms, developers integrate arifOS by calling the `judge_output` function with their LLM's output and the original query [62]. For example, after obtaining a raw answer from GPT-4 or Claude, one passes it to `apex_prime.judge_output(query, response, lane="HARD")` – arifOS then simulates the entire 000–999 pipeline on that content (Sense through Seal) and returns a structured result [62] [63]. The returned object includes the final verdict status, the possibly refined output (if PARTIAL, it will have inserted warnings or adjusted text), and a host of audit data like which floors were checked, metrics, and an **audit hash** [64] [65]. This allows external applications to programmatically **enforce the constitutional verdict** – e.g. only deliver answers that are "SEAL" (fully approved) or "PARTIAL" (with caveats), while handling VOID/HOLD by notifying a human or providing a refusal message. The **MCP server** can orchestrate this flow in real-time for chat applications or agent loops. Indeed, the repository includes reference UIs: a "raw" interface vs a "governed" interface that uses the MCP – in demonstrations the governed UI would show emoji labels like (Seal) or ⚠ (Partial) next to answers and block or modify outputs that failed checks [66] [67]. This proves the system can run live, not just on paper. The MCP infrastructure also supports **multi-agent** arrangements: e.g. spinning up subordinate agents for specialized tasks during 000_VOID (the code supports agent spawning and external tool calls [68] [69]). These agents all ultimately report back to the constitutional core for verdicts. Logging is centralized via the Cooling Ledger (see below). In short, the MCP layer makes arifOS into a *service* – whether via a Python API, a CLI (`demo_arifos_power.py`), or a web UI – that sits between user queries and model outputs to guarantee governance.

- **Cooling Ledger & Vault-999 (Tamper-Proof Memory):** A cornerstone of arifOS's runtime is that every final decision is recorded in an **append-only ledger** for accountability. The **Cooling Ledger** is the real-time audit log (stage 999) where each sealed response is logged with all relevant metadata [70]. This includes the prompt, the raw model answer, the final governed answer (which may be the same or modified), the verdict (Seal/Partial/etc.), all floor metric values, any warnings, votes from the W@W organs, and @EYE sentinel flags [70]. Because the ledger is meant to be tamper-proof, arifOS v47.1 introduced an **immutable hash chain**: every new entry's hash links to the previous, forming a verifiable chain of custody [9] [71]. This means any attempt to alter the log would break the hash continuity, providing cryptographic proof of integrity. Periodically or for critical outputs, entries can also be written to **Vault-999**, which is the long-term secure storage (conceptually, "vault" of approved knowledge and constitutional records) [72] [32]. Vault-999 stores the **sealed artifacts** – for example, important answers or decisions that have been ratified – and snapshots of the constitution itself (so one can always audit which version of the law was in effect for a given decision) [73]. Together, the cooling ledger and Vault-999 fulfill Floor F6 Amanah (trust and integrity) and provide a basis for external auditors or regulators to verify the AI's behavior. In practical use, one might connect this ledger to a monitoring dashboard or even a blockchain for decentralized verification. (The design in v48 uses a simple SHA-256 hash chain ledger, which is conceptually similar to an append-only log or blockchain block list for AI outputs [9].) By providing **100% reconstructible audit trails**, arifOS can answer the "why did the AI say that?" question for every response [74] [75].

- **Fail-Closed Execution & Sandbox:** arifOS is engineered to *fail safe*. Any deviation from expected parameters leads to a controlled shutdown of the response (VOID) rather than undefined behavior. For example, if a required check is missing or a law file is inconsistent, the runtime contract demands an immediate VOID refusal [47] [50]. Hard floors are tagged in code as `failure_action: VOID` – meaning if any such check fails, the code will not generate an answer [76]. This philosophy extends to the execution environment: arifOS can be deployed in a **containerized sandbox** so that even if the AI agent writes code or executes tools (in 000_VOID stage), those actions are isolated. In fact, 000 stage capabilities include "Docker isolation – safety through containerization" [77] [78]. The agent can be allowed to create and run code, but only inside a constrained Docker container that cannot affect the host system or network except as permitted. If something goes wrong (e.g. infinite loop or unexpected output), the container can be frozen or destroyed without harm – a **fail-closed design** at the infrastructure level. This is complemented by timeouts and resource caps: v47.1 added a **Settlement Policy** with hard timeouts (1.5s for model, etc.) and fallback behaviors [79]. In other words, if any sub-process doesn't respond in time, the system doesn't hang or output partial junk – it triggers a safe failure (perhaps an 888_HOLD or refusal). Such measures ensure that arifOS sessions don't free-run beyond their constitutional leash.

- **MCP Server & Human Oversight:** The MCP interface is extensible – developers can integrate human-in-the-loop oversight at crucial junctures. For instance, when an **888_HOLD** verdict occurs (meaning the AI has determined it should not finalize an answer without clarification or approval [35]), the MCP could notify a human moderator through a dashboard or messaging queue. The human can then either provide the needed input or override the AI's hold with a **SEAL token** if they decide to proceed. arifOS is designed to recognize a valid **HUMAN_SEAL_TOKEN** to bypass certain protections in exceptional cases (for example, `/fag` mode will accept a human override token to access a normally forbidden file path, simulating “root” permission given by a person) [80] [81]. This layered design allows **human ratification** for anything truly sensitive. In terms of interface, one could build a simple web panel that lists pending HOLD cases or high-stakes queries, where a human can click “Approve” or “Adjust and Continue.” Although such a UI isn’t part of the open-source release, the pieces are in place (the verdict system, the ledger, and the token mechanism) to facilitate it. In demos, Arif (the creator) has acted as the human failsafe – e.g. some modes explicitly log that certain actions “require human SEAL,” ensuring a person is in the loop for irreversible decisions [16] [82].

Complementary Tools & Extensions for arifOS

Beyond the core components, there are complementary open-source tools and best practices that augment arifOS’s governed execution:

- **Output Validators (JSON/XML Schema Enforcement):** One way to further **guardrail an AI agent** is to enforce that its outputs conform to a strict schema or format. Open-source packages like *Guardrails AI* (by Shreya Rajpal) allow developers to define JSON/XML schemas for LLM outputs and automatically validate and correct the model’s responses. This pairs well with arifOS: for example, after arifOS seals an answer, a schema validator could double-check that the answer doesn’t contain disallowed content or formatting (an extra layer for Floor F3/F8 related to clarity and consistency). If the answer fails validation, it can be sent back into a SABAR-style revision loop. In essence, such validators act as mini-“floors” for structural correctness. arifOS already measures semantic metrics for truth and clarity; adding a JSON schema check (for tools requiring structured output) can prevent subtle errors. This approach – **structured output validation with automatic repair** – complements arifOS’s qualitative floors with quantitative format guarantees. It echoes the same philosophy: don’t trust the raw model, verify everything against an explicit specification.

- **Cryptographic Logging & Ledgers:** arifOS’s Vault-999 ledger can integrate or draw inspiration from robust ledger solutions. For instance, using an append-only log service like **Google Trillian** or a blockchain-based ledger (e.g. Ethereum or Hyperledger Fabric) could provide decentralized verification of the AI’s audit trail. In the current design, each response log is hashed and linked (SHA-256 chain) [9] – this is essentially a lightweight blockchain. Adopting an existing ledger framework could bring features like distributed consensus on the log’s state or public auditability. For example, one could publish the hash of each Vault-999 entry on a public blockchain for transparency. Additionally, secure logging tools (like **immudb** or AWS QLDB) offer tamper-evident databases that could store arifOS transcripts. These integrations would bolster Floor F6 (Integrity) by making it practically impossible to alter or forge the AI’s memory without detection. In high-stakes deployments (finance, legal), such cryptographic proof of an AI’s compliance log may be invaluable for audits or regulatory requirements [83] [84].

- **Human-in-the-Loop Ratification Interfaces:** To fully leverage the 888_HOLD mechanism and similar features, a user-friendly interface for human review is vital. One can envision a dashboard that shows the AI’s draft output alongside the flagged risks that triggered a HOLD. The human overseer can then choose to approve the output, ask the AI for clarification, or instruct it to adjust and try again. This is akin to a **“moderation queue”** for AI responses. Some open-source projects are exploring this space (for example, integrating with chat platforms where a moderator must press a button before an AI's response is actually delivered to the end-user). For arifOS, a simple implementation could use a web app (possibly the provided Gradio UI) where HOLD responses are not shown to the end-user until a supervisor enters a decision. The constitutional floors F1–F9 provide clear rationale for holds (e.g. *"Tri-Witness consensus not met – please confirm sources"* or *"Possible ethical ambiguity – human review required"*). By presenting those reasons in the UI, the human can quickly understand the context. This **HITL (Human-In-The-Loop)** layer aligns with arifOS's design that ultimately, for ambiguous cases, **a human conscience is the highest authority**. It ensures that arifOS remains a *tool for humans* to make safe decisions, not an autonomous judge on matters beyond its scope.

- **Fail-Safe Execution Environments:** As AI agents become more autonomous (writing code, calling APIs), containing their execution is critical. arifOS already advocates containerization (Docker) and virtualization at the 000-VOID stage for any "unbounded" exploration [77]. The community can further augment this with things like **seccomp sandboxing** or language-based sandboxes. For example, if the AI writes and executes Python code, one could run that code in a Firecracker microVM or with Linux seccomp filters to limit syscalls (no network access, no disk deletes, etc.). This way, even if the AI attempted an illicit action, the OS-level sandbox would prevent it – providing a **second line of defense** under Floor F6 (no manipulation/harm). Another approach is **restricted execution frameworks**: e.g. using a library to evaluate the AI-written code in a constrained environment (like Pyodide for sandboxed Python, or timeouts that kill the process if it tries something fishy). The design principle is "**fail closed** by default" – anything not explicitly allowed should be denied. arifOS's constitutional Floor system is essentially a software fail-safe; pairing it with system-level fail-safes creates defense in depth. In practice, the provided Docker setup (the repo includes a `docker-compose.yml` for arifOS) is configured to have strict resource limits and no external internet unless a tool explicitly allows it [77] [78]. This ensures that, for example, if the agent somehow tried to escalate privileges or access the host file system, it would be confined to its container (and likely trip a Floor F6 or F1 check in the process).

- **Integration with AI Frameworks:** arifOS is designed to be framework-agnostic, and indeed it has been tested as a layer on top of popular AI orchestration frameworks. Version 35.1.0 added support for the "Big 3" Python AI frameworks – **LangChain, LlamaIndex, and AutoGen** – covering chain-of-thought pipelines, retrieval-augmented generation, and multi-agent systems [85] [86]. This means one can plug arifOS in as a guardrail: e.g. wrapping LangChain's outputs through arifOS's `judge_output` before finalizing each step. AutoGen (multi-agent) can use arifOS to govern conversations among agents (the W@W federation organs map well to distinct agent roles, like an evidence-gathering agent for @GEOX, a reasoning agent for @RIF, etc. as seen in the **AGI Builder Skills** package [87] [88]). The open-source **AGI-Builder** toolkit for Claude, for instance, uses five organ skills (@WELL, @RIF, @WEALTH, @GEOX, @PROMPT) orchestrated by a router skill – exactly mirroring arifOS's constitutional organs and floors [87] [89]. That package demonstrates how complementary software can implement arifOS principles on top of an AI API: it routes tasks to specialized tools and triggers **SABAR repair loops** whenever a Floor metric falls below threshold [90] [31]. We can expect further integrations or libraries that make it one-click to apply arifOS governance to any LLM. For example, a **LangChain Constitutional Guard** could become available, using arifOS's `constitutional_floors.json` as a policy and enforcing it during chain execution. Such community-driven extensions will make it easier to adopt arifOS v48 in diverse AI systems without reinventing the wheel.

Sources: The description above consolidates information from the arifOS v45–v48 artifacts, public repository, and related documentation. Key references include the arifOS Codex (v35Ω) which enumerates the Floors, pipeline, verdicts, and actors [91] [50], the arifOS v47.1 README and release notes for recent enhancements (quantum governance, immutable ledger, etc.) [92] [9], the File Access Governor v45 paper for autonomy mode details [11] [12], and the GitHub repository which hosts the core runtime (stages 000–999, AAA engines, sentinel) and canonical spec files [93] [46]. Additionally, the *Deep Research Review of arifOS* and Arif's Medium article provide context on the thermodynamic principles (Δ, Ω, Ψ) and the motivation for each rule [94] [95]. All these components interlock to form **arifOS v48**, a multi-layered constitutional AI governance system that is concurrently a *code library*, a *formal rule set*, and a *philosophy of AI safety*. By combining built-in skills (000–999 pipeline, SABAR/Phoenix protocols), core governance engines (APEX/ARIF/ADAM and sentinel organs), and integrations from the broader open-source ecosystem, arifOS aims to deliver AI that is **"governed by law, not by prompt"** – a principle increasingly vital as we push toward more autonomous AI.

[11] [96] [9] [46]

---

1 22 23 24 26 32 34 35 36 37 38 44 47 48 49 50 51 70 72 73 76 91 arifOS codex
https://www.notion.so/723e222089944cd1a5ea3044b627acfe

2 3 4 5 6 7 57 58 96 gitforge.md
https://drive.google.com/file/d/1jTGYoClq3LX9yNMwmxZo6QOEMDWYoTVI

8 10 11 12 13 14 15 16 17 18 19 20 21 25 80 81 82 arifOS File Access Governance (FAG) Artifact v45.0.3.pdf
https://drive.google.com/file/d/1MEjeHX7xcDice8J0V4m_FqwOutVccQLQ

9 39 40 41 42 43 45 46 52 53 56 59 60 61 62 63 64 65 68 69 71 74 75 77 78 79 92 93 GitHub - ariffazil/arifOS: ArifOS — ΔΩΨ-governed constitutional kernel for AI agents.
https://github.com/ariffazil/arifOS

27 28 66 67 arifOS DEMO.pdf
https://drive.google.com/file/d/1y4r8lQUGLAHunyX0b2wTXYaJd4qm2kWt

29 30 SABAR Protocol v1.2 Analysis
https://www.notion.so/26103e02d04580109d5eed6c4e190cdb

31 87 88 89 90 AGI Builder Skills Package - README.txt
https://drive.google.com/file/d/1cMcQG_lK7_CYlOjHnED6mVyKE_RHPvVl

33 85 86 95 What If AI Couldn’t Hallucinate?arifOS: A Thermodynamic Constitution | by ARIF FAZIL (arifOS) | Dec, 2025 | Medium
https://medium.com/@arifbfazil/what-if-ai-couldnt-hallucinate-arifos-a-thermodynamic-constitution-204a8a9bb953

54 AGENTS.md – An Open Format for Guiding AI Coding Agents with ...
https://www.blog.brightcoding.dev/2025/09/21/agents-md-an-open-format-for-guiding-ai-coding-agents-with-project-instructions/

55 Context Engineering for Developers: The Complete Guide | Faros AI
https://www.faros.ai/blog/context-engineering-for-developers

83 84 ArifOS – A Constitutional Physics Layer for Governed AI.pdf
https://drive.google.com/file/d/1xL3w1Us9q7O7UmNrqcMada0gf8Kk5S8M

94 Deep Research Review of arifOS.pdf
https://drive.google.com/file/d/1WiBFpxPKbGXtXyoGsDgrh2kD7Haa17OF