ChatGPT

arifOS Memory Stack – Master Canon (v36.3Ω)

Version: 36.3Ω
Role: Constitutional Law – Memory Architecture and Retention
Status: Active (Upgraded from v36.3Ω to v37)
Scope: Defines the structure of system memory bands, retention policies, and amendment algorithms (Phoenix-72) as inviolable law.

Preamble

This document constitutes the **Memory Stack Canon** for arifOS v36.3Ω. It codifies the fundamental **laws of memory** – how the system organizes context, retains history, and evolves its knowledge via amendments. All subordinate specifications, code, and tests must adhere to these principles. The Memory Stack is governed thermodynamically (inspired by entropy and control theory) to ensure stability, consistency, and integrity of the AI's memory across runs. We establish herein the six-band MemoryContext model, the Phoenix-72 feedback amendment policy, strict data retention tiers, vector embedding standards, and integrity safeguards. This Canon has highest authority (L1 Law), superseding any conflicting implementation detail.

MemoryContext Structure – The Six Bands of Memory

The system **MUST** construct exactly one **MemoryContext** object per run. This MemoryContext is partitioned into six distinct **memory bands**, each with a specific role and authority level. The bands are isolated yet collectively constitute the runtime context:

1. **Ω-Env Band (Environment Metadata):** *Immutable runtime metadata.* This read-only band contains deployment context and run-specific metadata such as current timestamp, organization or tenant ID, environment risk profile, and other execution parameters. It provides situational context but **cannot** be altered during a run. (Authority: *Informational* – it frames the query but does not override any law.)

2. **Vault Band (Constitutional Memory – L0):** *The inviolable Constitution.* This band holds the entire constitutional knowledge base: the **Floors** (thresholds/limits), **Physics** constants, and all active **Laws** and amendments. It is loaded from the secure Vault (`vault_999/constitution.json`) at runtime. The Vault Band has **absolute authority** – its contents (the "Constitution") override any other memory. This band is read-only during normal operation; changes occur only through controlled **Phoenix-72** amendment cycles (with proper signing and audit). No ephemeral context or external input may override the Vault Band's law.

3. **Ledger Band (Historical Ledger Context):** *Recent system history.* This band provides a time-windowed slice of the **Cooling Ledger** – a chronological log of recent interactions, decisions, and

outcomes. It typically includes the last ~7 days of entries or a fixed number (e.g. 10k) of recent events. The Ledger Band's authority is **historical**: it represents ground-truth records of what has happened (e.g. recent user queries, system actions, verdicts), informing current decisions with past context. It is append-only (prior entries are sealed with cryptographic hashes) and pruned/rotated according to retention policy (see Retention tiers below). The Ledger Band cannot contradict the Vault (law) but serves as evidence for reasoning and for Phoenix-72 analysis.

4. **Active Stream Band (Volatile Run Memory):** *Current run transient context.* This band captures the live **Active Stream** of the ongoing interaction within the current run: user input, the system's draft response, intermediate reasoning (Chain-of-Thought), and live audit results. It is essentially the working memory for one cycle of input→draft→audit. The Active Stream is **volatile** and writeable during the run – it updates in real-time as the system processes the query. However, it is not considered stable or authoritative memory until the run completes. Once sealed (e.g. the answer is finalized and logged), relevant pieces of this band may be written into the Ledger (thus graduating to historical memory). Until sealed, the Active Stream is ephemeral and isolated, preventing unverified interim thoughts from polluting long-term memory.

5. **Vector Witness Band (Retrieved Context – RAG):** *External soft evidence.* This band contains any **retrieval-augmented generation (RAG) data** pulled in via vector search – e.g. documents, knowledge base articles, or other reference texts relevant to the query. These are provided as **witness** evidence to inform the system, but are not verified law. The authority of this band is *soft*: it can influence the answer but must always be regarded with uncertainty. **Critically, the Vector Witness Band MUST remain isolated from the Vault Band's constitutional content** to prevent "hallucination by proximity." In practice, this means the system should clearly distinguish or separate any external info from the core laws/ledger in prompts or memory – the model should not confuse witness data with guaranteed facts. The witness content is unsigned and untrusted by default; it augments knowledge but cannot override the Vault or Ledger truths.

6. **Void Band (Diagnostics & Scars):** *Errors, anomalies, and proposed constraints.* This band collects diagnostic information from the run – for example, any rule violations, errors/exceptions, or anomalous outcomes that occurred. Crucially, the Void Band is also where **Scar proposals** are recorded. A "Scar" is a negative constraint or cautionary rule the system proposes in response to a serious failure (e.g. a VOID verdict, SABAR event, or policy breach). When a run triggers a floor failure or other critical issue, the system may generate a Scar proposal (including context like which floor was broken, a description of the issue, severity, etc.) and store it in the Void band for later review. This band has no influence on the user-facing output; it is purely for internal governance and will feed into the Phoenix-72 process. Content in the Void Band is transient (scars will be logged to a proposal file for audit) and **carries no authority** at runtime – it is an input to future law amendments, not law itself yet.

Session Band (Optional): If multi-turn conversational sessions are enabled, an additional **Session Memory Band** MAY be maintained (keyed by session or user ID). This would contain the dialog history and context across multiple turns for that session, separate from the global ledger. Session memory is ephemeral to the session and not persisted indefinitely (it may expire or be cleared when the session ends). The Session Band, if used, must remain subordinate to the Vault and subject to the same witness isolation rules (i.e. a user's past utterances are context but do not override constitutional laws). This band is optional and not counted among the core six; it exists to support chat continuity on a per-session basis.

Phoenix-72 Amendment Policy – Damped Feedback Control

To prevent oscillation or “flapping” of governance floors (rapid toggling of rules on/off) and to adapt smoothly to system drift, arifOS employs the **Phoenix-72** amendment policy. Phoenix-72 is a control-theoretic approach (inspired by PID controllers) to adjust constitutional parameters (like floor thresholds or adding scars) based on observed failures, rather than immediate one-off reactions. All floor adjustments and new law incorporations **must** follow this governed process:

- **Damped Feedback Algorithm:** Instead of binary triggers, adjustments are computed with a weighted feedback formula. Let recent failure **Frequency** (how often a particular floor or rule was violated in the Phoenix observation window, e.g. 72 hours) be F, and failure **Severity** (how critical or how far beyond tolerance the failures were) be S. We define a **pressure** metric P as a linear combination:

$$P = w_1 F + w_2 S ,$$

where w_1 and w_2 are tuning weights favoring frequency vs. severity. This P value reflects the system’s impetus to change a floor value or introduce an amendment: a higher pressure means the law is insufficient (too lenient) and needs strengthening; negative or low pressure might suggest the law is too strict or stable.

- **Amendment Decision Logic:** The Phoenix-72 controller uses P to determine adjustments in a **damped** manner. Rather than jumping to new values, it proposes a gradual change (Δ) to a floor threshold or enacts a new scar pattern, proportional to the pressure (similar to a proportional controller, with potential integrative memory over 72h). The goal is to reach a new equilibrium without overshoot. The exact mapping from P to a Δ adjustment is defined in the Phoenix spec (typically small increments scaled by a constant factor).

- **Constraints (Law of Amendment):** The following constraints are **hard-coded in law** for any Phoenix-72 amendment cycle:

- **Deadband (Humility):** If the calculated adjustment magnitude is below a small epsilon (e.g. < 0.005), **no change is made**. The system remains humble in face of minimal deviations, avoiding noise or trivial tweaks. Minor fluctuations in performance do not trigger amendments.
- **Hysteresis Cooldown (Sabar – Patience):** After a floor or law has been amended, a cooldown period of 24 hours (minimum) is enforced during which no further adjustment to that same parameter is allowed. If an amendment was applied less than 24h ago, the affected floor is **locked** and will not be amended again until the period elapses. This prevents rapid back-to-back changes (oscillation) and gives the system time to observe the effect of one change before another. Essentially, each floor amendment carries a grace period where Phoenix-72 will hold its hand (patience).
- **Safety Cap:** No single adjustment may exceed **±0.05** in magnitude for continuous parameters (or analogous small steps for discrete settings) per cycle. This caps the impact of any one Phoenix-72 run, ensuring gradual evolution. Even if pressure is extremely high, the system will ratchet the floor in multiple small steps over multiple cycles rather than one large jump. This protects stability and prevents overshooting the optimal value.

- *(All amendments must also be logged as formal Amendment records in the Vault's constitution, including timestamp, Phoenix cycle ID, details of changes, and supporting evidence. Phoenix-72 operates under Tri-Witness oversight – meaning it considers Vault law, Ledger data, and external witness metrics in concert to justify changes.)*

In summary, **no direct manual or automatic tweaks to governance floors are allowed outside Phoenix-72.** The law mandates that all changes flow through this PID-like controller with the above constraints. By doing so, arifOS maintains a stable "metabolism," adjusting slowly and deliberately to new information or drifts in model behavior, rather than reacting impulsively. This prevents chaotic policy shifts and ensures any amendment is well-justified and documented.

Retention Policy – Hot, Warm, Cold Memory Tiers

Unbounded growth of logs or context is forbidden. The Memory Stack follows a strict **three-tier retention strategy** to manage the Cooling Ledger and related memory artifacts. This policy ensures recent information is readily available, while older data is archived or summarized, and an immutable proof-of-history is maintained indefinitely. The tiers are defined as follows:

- **Hot Tier (Active Ledger):** The Hot tier consists of the live cooling ledger file (e.g. `runtime/cooling_ledger.jsonl`) containing the most recent entries. Only the last **7 days** of activity or the last **10,000 entries** (whichever limit is reached first) are kept in the hot ledger. This ensures the in-memory context (Ledger Band) deals with a bounded, relevant window of history. As new entries come in beyond these limits, older entries are **rotated out** to the Warm tier. The Hot ledger is used directly for real-time context and Phoenix-72 scar scanning.

- **Warm Tier (Recent Archive):** The Warm tier stores recent history that has aged out of Hot. Ledger data older than ~7 days is compressed and archived in monthly files (e.g. `archive/cooling_ledger_2025-12.jsonl.gz` for December 2025). Each archive file contains one calendar month of JSONL entries and is kept for at least **1 year**. These archives are not loaded in active memory but can be consulted for deep audits, retrospective analysis, or offline training data extraction. After 1 year (or a policy-defined retention period), warm archives may be purged or moved to long-term storage, as long as their cryptographic anchors remain (see Cold tier). **No ledger data should reside indefinitely in Hot/Warm storage beyond the retention window.** This prevents infinite file growth and reduces storage burden while preserving medium-term audit trail.

- **Cold Tier (Perpetual Anchors & Proofs):** The Cold tier is an immutable skeleton of history retained forever in the Vault. Instead of full logs, it holds **cryptographic anchors** and essential receipts that prove history without storing it verbatim. For each rotated ledger file or significant event, a cryptographic **tip** or hash (e.g. the final hash of a ledger segment) is stored under `vault_999/anchors/`, possibly along with a high-level summary or "EUREKA receipt" of notable outcomes. These anchors create an immutable chain of custody for the system's knowledge: even if the raw logs are archived or deleted after 1 year, the hash chain can always verify that those logs existed and were unaltered up to that point. The Cold tier thus provides eternal **proof of history** and supports compliance/audit needs, without needing to keep full detailed records forever. *(Example: a* `head_state.json` *file may record the last entry hash and timestamp whenever the ledger rotates, to be used as the first link when a new ledger begins, ensuring continuity.)*

Lawful Requirements: Under this retention law, **infinite append-only files are strictly disallowed.** The system must implement rotation and archiving such that no single JSONL grows unbounded. The transitions from Hot→Warm→Cold are to be automated and atomic (e.g. on rotation, ensure the last hash of the old file is stored in Cold before beginning a fresh hot file). All three tiers together guarantee that memory is manageable in the short term, useful in the medium term, and verifiable in the long term.

Vector Embedding Standards – Witness and Scars Indices

The Memory Stack leverages vector embeddings to store and retrieve contextual information. All embeddings and vector searches in arifOS v37 **must conform to these standards** to ensure consistency, performance, and security:

- **Embedding Model:** The system shall use the **Sentence-Transformers** `paraphrase-multilingual-MiniLM-L12-v2` **model** for text embeddings. This model is chosen for its multilingual capability (Nusantara-ready, supporting languages of the Malay archipelago and beyond) and efficiency (small enough for CPU inference). It is hosted **locally** – no external API calls or dependencies (like online services) are required or allowed for embedding generation. This guarantees data sovereignty and low-latency memory lookups. All textual data destined for vector storage (e.g. knowledge snippets, past queries, scar texts) will be embedded via this model to a 384-dimensional vector (as defined by the MiniLM architecture).

- **Vector Store Implementation:** Embeddings are stored and queried using on-premise tools: a combination of **SQLite** (for metadata and exact filtering) and **FAISS** (Facebook AI Similarity Search for fast vector similarity). The system will maintain local index files; no cloud vector database (e.g. Pinecone) is used, in line with offline deployment requirements. Each vector index is persisted on disk and updated as new data comes in, with appropriate locking or transaction safety to avoid corruption.

- **Index Segregation – Scars vs. Witness:** There must be **two distinct vector indices** serving different purposes, and they should never be conflated:

- **Scars Index (Negative Constraints):** A global vector index containing embeddings of all **"Scar" entries**, which represent forbidden or highly sensitive patterns the system has learned to avoid. Each scar in this index corresponds to a signed negative rule or problematic query/response that was identified (often via Phoenix-72 or human review) and canonized as something to watch out for. Because scars are effectively part of the system's protective laws (negative constraints), this index is **global** (applies across all tenants/contexts) and each entry **must be cryptographically signed by the system's key** to ensure authenticity (see Security below). The Scars index is consulted whenever new input is processed, to gauge similarity to known dangerous scenarios and apply immediate preventative measures if necessary (like triggering high scrutiny or refusal). Under no circumstances may an unsigned or external piece of data be inserted into the Scars index.
- **Witness Index (Knowledge Base):** One or more vector indices used for general knowledge retrieval – essentially storing embeddings of various documents, articles, or prior interactions that can serve as **witness evidence** for answering queries. These indices are typically **per-tenant or per-domain**, containing content relevant to that context (and isolated from other tenants' data for privacy). Unlike scars, witness entries are **unsigned** and are not assumed to be absolutely true; they are auxiliary information. The Witness index provides soft context (as the Vector Witness Band), and its entries

can be updated or removed as the knowledge base evolves. Importantly, any content in the witness index remains at a *lower authority* level: it can help formulate answers but cannot on its own establish truth against the Vault's laws or override them.

By separating these indices, the system ensures that **negative constraints (Scars) and factual references (Witnesses) never mix**. This segregation is law: it prevents a scenario where a piece of unverified external text might be treated as a hard constraint or where a scar (which might contain a description of disallowed content) accidentally influences answer generation as if it were just another document. Each index has its access controls and usage policies aligned with its purpose – the Scars index being security-critical and global, the Witness indices being informational and scoped.

System Integrity and Security Guarantees

Finally, the Memory Stack Canon mandates specific measures to preserve integrity of data and prevent malicious or accidental compromise of the memory system:

- **Hash-Chain Continuity (No Drift):** The Cooling Ledger must maintain an unbroken **hash-chain** across rotations. Every entry appended to the ledger carries a cryptographic hash link to its predecessor (e.g. a SHA-256 of the previous entry or cumulative chain). When the Hot ledger file rotates (e.g. monthly), the **final hash** of that segment shall be written to a persistent **head state** (for example, in `vault_999/anchors/head_state.json`). The next ledger file must incorporate this previous hash as its starting `prev_hash`, chaining the files together. This law ensures that even though logs are segmented for retention, one can always detect any removal or tampering with historical records. "Hash-chain drift" – losing the lineage of hashes due to improper rotation – is not allowed. The system's rotation logic must explicitly carry forward the last known hash and any necessary state so the ledger remains **cryptographically verifiable** end-to-end.

- **Witness Content vs. Signed Law (No Poisoning):** The system shall treat any **unsigned** memory content as untrusted **witness** data. Only data that is **cryptographically signed by the arifOS System Key** (the private key governing the instance) can be considered part of the Vault law or authoritative constraints. In practice, this means all official amendments (floor changes, new scars) are signed and recorded in the constitution. If someone or something injects content into the system (for example, a user provides a piece of text that looks like a rule, or an external source suggests a policy), it **must not** be treated as law unless it goes through the proper Phoenix-72 review and gets signed. **Witness poisoning** – a scenario where malicious input could trick the AI by sitting near authoritative data – is prevented by this signature requirement and by the band isolation described above. Unsigned scars or rules are by definition **invalid**; they remain in the Witness category until vetted and signed. The system should downgrade or ignore any purported "law" that lacks a valid signature, thereby upholding the sanctity of the Vault.

- **Constitutional Amendments Audit:** Every change to the memory constitution (new law, amendment, or removed law) must be logged in the Vault's amendments list with full details (timestamp, reason, evidence, author). This ensures traceability. While this is a procedural point, it is elevated to Canon: **no silent changes** to memory structure or thresholds can occur. All must pass through official channels (Phoenix-72, admin action) with audit trail.

- **Isolation of Sensitive Data:** Bands like the Void (with scars/diagnostics) and Scars Index may contain sensitive or toxic content (since they record things to avoid or bad examples). The law requires that this content **never be directly exposed to the user or to parts of the model that generate user-facing text**, except in controlled analysis contexts. It should influence decisions (by informing the model what to avoid) but should not be regurgitated. This protects users from seeing disallowed content and prevents the model from getting inadvertently "infected" by the very things it's avoiding.

By these laws, the arifOS Memory Stack (v36.3Ω → v37) is defined and governed. All subordinate specifications (contracts) must reflect these principles in precise detail (via JSON schemas and logic), and all code must implement them faithfully. Tests must be devised to ensure no violation of these memory laws is possible. This Canon stands as the Constitution of Memory for arifOS – durable, explicit, and optimized for the next generation of trustworthy AI operation.

JSON Schemas to Generate (Step 2):

- `MemoryContext.schema.json` – Schema defining the structure of the MemoryContext object, including all six bands (and optional Session band), and the expected content/types for each band.
- `CoolingLedgerEntry_v36.schema.json` – Schema for a Cooling Ledger entry (v36Ω format), including fields like timestamp, hashes (`prev_hash`), metrics, verdict, etc., to enforce the log structure.
- `LedgerHeadState.schema.json` – Schema for the head state anchor file (e.g. containing the last hash of a ledger segment, timestamp, and any chain metadata required to link ledger files).
- `Phoenix72Amendment.schema.json` – Schema for a Phoenix-72 amendment record, detailing how floor updates or new laws are represented (ID, applied_at, details of changes, evidence links, signature).
- `ScarRecord.schema.json` – Schema for a Scar definition (negative constraint entry), including fields for the scar text/pattern, description, severity, origin (ledger reference), and the required digital signature field to mark it as verified.
