Civilization Memory Architecture v0.00.01–v0.00.06: Auditable Multi-System Memory from Episodic Binding to Procedural Skills and Multi-Scale Graphs

Yike Wang

This manuscript is a technical preprint and has not undergone peer review. It documents the versioned implementation and evidence package available with this release.

***Abstract*—Large language model memory is often implemented as an undifferentiated text store, making it difficult to distinguish transient state, specific events, generalized knowledge, executable strategies, and policy decisions. This technical report presents the Civilization Memory Architecture as a versioned model–system architecture developed from v0.00.01 Sun through v0.00.06 Rosette. It provides a complete public map of Stages44–150 while referring to the companion study for the earlier hidden-state experiments. The version line combines a frozen model runtime, four typed memory systems, deterministic retrieval, hippocampal-style episode binding and replay, approval-gated episodic-to-semantic consolidation, execution-trace-to-procedure formation, atomic conflict-aware context assembly, and a multi-scale graph view. Memory cells are never silently overwritten during conflict handling: competing facts and control decisions remain separately addressable and linked. Mutations and policy reads emit trace events, while validated snapshots use canonical serialization, integrity hashes, and fail-closed recovery.**

Evaluation separates model-path diagnostics from memory-structure validation. In a fresh WSL/CUDA reproduction, 40 Orion, 20 Trifid, and 29 Lagoon–Eagle–Rosette tests passed. A frozen Qwen3-0.6B diagnostic produced a $1 \times 1 \times 1024$ memory vector, nonzero full-path deltas (7.793 and 4.919), zero deltas under the no-memory intervention, and no trainable base-model parameters or gradients. A controlled five-node graph fixture covered three scales and three adjacency types, with deterministic traversal reaching every node. The report also traces rejected task-recovery, incomplete-evidence, conflict, context-budget, and recovery cases into explicit constraints. These results establish executable contracts, provenance, intervention boundaries, and recovery behavior. They do not establish improved open-domain accuracy, autonomous lifelong learning, production reliability, or human-equivalent memory.

***Impact Statement*—Auditable memory is a prerequisite for deploying systems that retain and reuse experience. The proposed architecture makes the origin, type, lifetime, conflict status, and approval state of remembered information explicit. This supports inspection, rollback, and policy enforcement before memory reaches a model context. The contribution is an engineering and experimental foundation, not a claim that machine memory reproduces human cognition. The preserved failure cases and strict scope boundaries are intended to reduce the risk of mistaking a passing smoke fixture for real-world reliability.**

***Index Terms*—episodic memory, semantic memory, procedural memory, memory consolidation, memory-augmented language models, graph memory, auditability.**

Yike Wang is with Hydite AI Research, Haokir Inc., Colorado Springs, CO 80918, USA (e-mail: yikewang@research.hydite.com; ORCID: 0009-0009-1832-5421).

AI tools were used for drafting assistance, language refinement, LaTeX organization, and consistency checking; all technical claims, experiments, results, and final responsibility remain with the author.

I. Introduction

Persistent context changes an artificial agent from a stateless predictor into a system whose earlier observations can affect later decisions. A flat store can provide recall, but it does not state whether an item is a short-lived working value, a specific event, an abstraction over events, or an action policy. It also obscures when an abstraction was approved, which episodes support it, whether contradictory evidence exists, and whether a retrieved item was actually routed into model computation.

Biological accounts motivate complementary fast and slow learning systems without requiring a literal neural simulation [1], [2]. In machine learning, external retrieval, memory networks, episodic control, and graph-based reasoning provide complementary mechanisms [3], [4], [5], [6]. The unresolved systems question is how to combine these mechanisms while retaining typed state, provenance, explicit policy boundaries, and failure artifacts.

This paper reports the design and executable validation of Civilization Memory Architecture. The work extends an earlier structured hidden-state architecture with a software memory substrate rather than replacing the language model [7]. Fig. 1 summarizes the six-version route. Sun provides a frozen-model and service baseline; Orion introduces the four-system kernel; Trifid rapidly binds episodes and replay evidence; Lagoon performs approval-gated semantic consolidation; Eagle derives procedural candidates from task traces; and Rosette assembles atomic context groups and a multi-scale graph.

The contributions are:

- A common cell/link/trace representation for working, episodic, semantic, and procedural memory, including deterministic retrieval and TTL behavior.
- A provenance-preserving path from rapid episode binding through replay and approved semantic consolidation, with explicit conflict review and preserve-both decisions.
- A trace-backed procedural memory path and a typed, bounded honeycomb graph projection over episode, schema, and control scales.

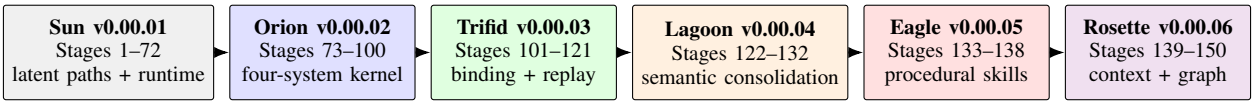

Fig. 1. Six-version implementation route. Each version preserves the prior contracts and adds one bounded responsibility.

- A fail-closed verification protocol joining unit tests, release gates, canonical snapshots, fresh CUDA diagnostics, artifact checks, and an evidence ledger that distinguishes historical results from the present reproduction.
- A complete public responsibility map for Stages44–150 that connects each implementation increment to its acceptance boundary without disclosing private deployment configuration or parameter artifacts.

The claims are intentionally structural. The reported evidence shows that the implemented contracts execute and that the memory path can be intervened on in a controlled model fixture. It does not show a general task-accuracy advantage over RAG, adapters, or other memory architectures.

The remainder of the report first formalizes version transitions and the public disclosure boundary. It then develops the six releases in causal order: Sun's runtime substrate, Orion's governed memory, Trifid's episode mechanism, Lagoon and Eagle's approved semantic/procedural transitions, and Rosette's context and graph views. A failure-driven analysis and complete stage map precede implementation, evaluation, results, limitations, and the evidence appendix.

II. Related Work

A. External and Differentiable Memory

Retrieval-augmented generation and retrieval-scaled language models condition generation on nonparametric evidence [3], [8]. Memory Networks and Neural Turing Machines instead expose trainable memory access mechanisms [4], [9], [10]. The present architecture is orthogonal to retrieval scale: it concentrates on typed lifecycle semantics, provenance, conflict preservation, and auditable handoff to a frozen decoder.

B. Complementary Learning and Episodic Control

Complementary learning systems theory separates rapid episodic acquisition from slower integration of structured knowledge [1], [2]. Neural Episodic Control demonstrates fast value reuse from episodic records [5], while HippoRAG combines knowledge graphs and retrieval with a neurobiologically inspired framing [11]. Trifid and Lagoon borrow the engineering intuition of fast binding followed by slower consolidation. They do not claim biological fidelity: replay scheduling, signatures, and approvals are deterministic software policies.

C. Agent Memory and Graph Structure

Generative Agents combine observation, reflection, planning, and memory retrieval in an interactive agent architecture [12]. Graph Networks and graph attention formalize relational computation over typed entities and edges [6], [13], [14]. Rosette differs from a learned graph network: its first implementation is a deterministic projection and traversal layer over validated memory links. No learned graph weights or performance advantage is claimed.

D. Frozen-Model Adaptation

Adapters, LoRA, and prefix tuning provide efficient alternatives to full-model fine-tuning [15], [16], [17]. The Sun baseline uses path-specific residuals as intervention surfaces while keeping Qwen3 frozen. The memory architecture reuses this surface to test whether a retrieved cell changes the hidden path, but the memory kernel itself remains pure Python and does not require model training.

III. Problem Formulation and Invariants

Let $\mathcal{S} = \{W, E, S, P\}$ denote working, episodic, semantic, and procedural memory. A memory cell is

$$c_i = (id_i, s_i, x_i, \bar{x}_i, o_i, p_i, u_i, t_i, d_i, k_i, m_i), \tag{1}$$

where $s_i \in \mathcal{S}$, x_i and $\bar{x}_i$ are content and summary, o_i is source provenance, $p_i = (q_i, r_i)$ contains confidence and importance, u_i contains creation/update times, t_i is an event index, d_i and k_i are decay and consolidation states, and m_i is typed metadata. A link $\ell_{ij} = (id_i, id_j, \tau, w, t, m)$ has type

$$\tau \in \{temporal, similarity, causal, task, procedure, replay, conflict\}. \tag{2}$$

The live store is the graph $G = (C, L)$ plus an append-only trace sequence $A = (a_1, \ldots, a_n)$ over write, read, link, update, expire, consolidate, and conflict events.

The implementation is governed by five invariants. First, a conflict operation adds evidence and links; it does not delete either fact. Second, a semantic or procedural write derived from prior evidence carries source identifiers and an explicit approval identifier. Third, an inactive working cell is excluded from default retrieval. Fourth, a control decision is never serialized as if it were factual evidence. Fifth, snapshot loading fails closed when the envelope, canonical hash, topology, or referenced cells disagree.

For query term set Q and cell term set X_i, Orion uses the deterministic score

$$R(c_i, Q) = \frac{|Q \cap X_i|}{\max(|Q|, 1)} + 0.15 I_s + 0.2 r_i + 0.2 q_i + 0.05 \sum_{\ell \sim i} w_\ell, \tag{3}$$

where $I_s = 1$ only when an explicitly requested memory system matches. Sorting uses stable identifiers to break ties. Equation (3) is a transparent baseline, not a learned relevance model.

An episode frame binds a scene, ordered source cells, entities, goal, action, outcome, cues, and temporal interval. Its consolidation signature is

$$\sigma(e) = (scene, entities, goal, action, outcome). \quad (4)$$

A candidate is admissible only when every source step has a replay link and at least the configured number of frames share σ. Approval produces an idempotent semantic write keyed by a stable fingerprint. This separates evidence gathering from state mutation.

IV. Versioned Transition Contract

The six releases are treated as a single versioned system rather than six independent feature lists. Let the release state at version v be

$$\mathcal{M}_v = (C_v, L_v, A_v, \Pi_v, \Omega_v), \quad (5)$$

where C_v, L_v, and A_v are cells, links, and audit events; Π_v is the explicit transition and admission policy; and Ω_v is the serializable operational envelope, including snapshots, manifests, and recovery rules. A version transition is admissible only when it adds a bounded capability while preserving the authoritative status of prior cells and links:

$$\mathcal{M}_{v+1} = \mathcal{M}_v \oplus \Delta_v, \qquad \mathrm{auth}(C_v, L_v) \text{ is unchanged by } \Delta_v. \quad (6)$$

Here "unchanged" does not prohibit a new link, decay-state annotation, or derived cell. It prohibits a transition from silently replacing a prior factual cell, erasing its provenance, or converting a control decision into a fact. The release gates test bounded instances of this contract; they do not constitute a formal proof over all future states.

Three cross-version rules organize the implementation. First, evidence is monotone: binding, replay, clustering, and skill candidacy may add derived objects but retain their supporting identifiers. Second, authority is explicit: semantic and procedural writes require an approval-bearing policy transition rather than a retrieval score alone. Third, a query-specific view is non-authoritative: Rosette packets and graphs may be rebuilt or rejected without mutating the underlying store. These rules make it possible to audit which transition introduced a new object and which policy admitted it to a model context.

Sun is the compatibility base for the frozen-model path and the runtime surface. Orion introduces typed cells and a policy boundary between local and global evidence. Trifid adds ordered episode frames without automatically consolidating them. Lagoon and Eagle turn complete evidence into semantic or procedural objects only through explicit approval. Rosette does not replace these stores with a graph database: it projects validated evidence into a bounded graph view. Consequently, a later graph traversal is explainable in terms of earlier cells, links, approvals, and context groups.

V. Architecture Overview and Disclosure Boundary

The implementation line has two connected planes. The neural/runtime plane originates in Sun and exposes a frozen-model context-vector interface. The memory plane begins with Orion's typed store, adds episode structure in Trifid, controlled semantic and procedural transitions in Lagoon and Eagle, and query-bounded graph views in Rosette. The planes meet at a narrow bridge: only selected factual memory items are serialized and encoded for the Adapter; control records remain available to the policy and audit layers but are not silently presented as facts.

This report publishes module responsibilities, typed object relationships, state-transition conditions, negative controls, aggregate test results, and sanitized artifact manifests. It does not publish host addresses, credentials, deployment tokens, private model locations, proprietary task payloads, trained parameter files, or operational configuration values. Implementation identifiers are included only where they are necessary to map a claim to a stage or artifact. This boundary permits technical scrutiny of the architecture without turning the paper into a deployment manual or a source release.

The stage granularity reflects failure isolation rather than independent scientific discoveries. Some stages introduce a data structure; others close a lifecycle gap, exercise a negative control, or freeze a release contract. The following sections therefore organize stages by architectural problem and explain the causal relation between them. A complete public Stage44–150 map is provided later in the paper.

VI. Public Object and Transition Semantics

The version names describe releases, but the technical unit that connects the releases is a guarded state transition. This section specifies that unit at the public interface level. It deliberately omits private deployment configuration and internal payloads while exposing enough structure to distinguish stored evidence, proposed abstractions, approved memory, control state, and query-specific views.

A. Five Authority Classes

Objects belong to five authority classes. *Observed evidence* consists of working or episodic cells and their source metadata. These objects report what entered the system; their presence is not a truth guarantee. *Bound evidence* consists of episode anchors, ordered source references, temporal links, and replay links. Binding adds organization without replacing the observations. *Proposals* are consolidation clusters, schema candidates, and skill candidates. A proposal can be inspected and rejected, but is not eligible as semantic or procedural memory. *Approved derived state* consists of semantic schemas and procedural skills produced by an explicit transition. Finally, *control and view objects* include conflict decisions, context packets, graph projections, traversal results, and recovery reports. They govern or describe access to facts but do not become facts by being colocated with them.

This distinction resolves an ambiguity common in memory systems: retrieval, selection, and consolidation are not synonyms. Retrieval observes an existing state; selection proposes a bounded subset; consolidation creates a new typed object only after its guards pass. Likewise, a conflict decision authorizes a handling mode but does not certify either conflicting statement as universally true.

TABLE I
CROSS-VERSION TRANSITION CONTRACTS. A RELEASE ADDS THE STATED BOUNDED CAPABILITY WITHOUT CHANGING THE AUTHORITY OF PRIOR FACTUAL EVIDENCE.

Transition	Added object or view	Required preservation rule	Release evidence
Sun → Orion	typed cells, session/global tiers	local session isolation; global read is explicit	Stage73–100
Orion → Trifid	episode frame and replay links	ordered source cells remain addressable	Stage101–121
Trifid → Lagoon	cluster and semantic schema	complete replay plus approval before derived write	Stage122–132
Lagoon → Eagle	trace-backed skill cell	success and failure traces remain linked	Stage133–138
Eagle → Rosette	packet and honeycomb graph	graph is a non-authoritative validated projection	Stage139–150

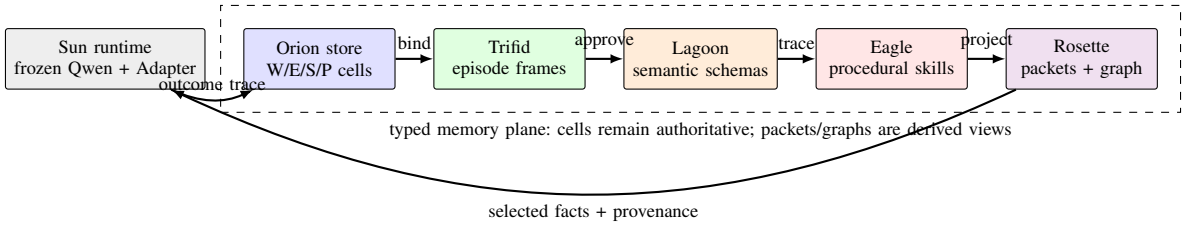

Fig. 2. Public end-to-end architecture. The model/runtime and memory planes meet through selected factual context and an auditable outcome trace.

TABLE II
VERSION AND RESPONSIBILITY MATRIX

Version	Stages	Primary responsibility	Persistent output	Explicit non-claim
Sun v0.00.01	1–72	latent paths, frozen backend, service	model/run artifacts	general task superiority
Orion v0.00.02	73–100	four memory systems and lifecycle	cells, links, traces, stores	learned retrieval
Trifid v0.00.03	101–121	rapid binding, replay, checkpoints	episode/replay evidence	biological simulation
Lagoon v0.00.04	122–132	episode-to-schema consolidation	schemas, conflicts, decisions	automatic truth resolution
Eagle v0.00.05	133–138	trace-to-procedure formation	skill and trace provenance	optimal policy learning
Rosette v0.00.06	139–150	atomic context and graph projection	packets, graphs, snapshots	learned graph reasoning

Let an object carry status

$$z \in \{observed, bound, replayed, candidate, approved, control, view\}. \tag{7}$$

The implementation does not encode this enumeration in one shared field; instead, memory type, consolidation metadata, links, and artifact schema jointly determine the status. The enumeration is therefore a public semantic model of the implemented contracts, not a claim that one hidden finite-state machine controls every module.

B. Guarded Mutation

For state $\mathcal{M}$, proposed transition T, evidence bundle X, and approval record u, define a guard vector

$$g(T, X, u) = (g_{type}, g_{source}, g_{complete}, g_{policy}, g_{identity}). \tag{8}$$

The components respectively check type compatibility, source existence, evidence completeness, policy authorization, and deterministic identity. A mutating transition is admitted only when all required components are true:

$$T(\mathcal{M}, X, u) = \begin{cases} \mathcal{M} \oplus \Delta_T, & \bigwedge_k g_k = 1, \\ \mathcal{M}, & \text{otherwise.} \end{cases} \tag{9}$$

The rejected branch may emit a failure result or audit event, but it must not publish a partial semantic cell, procedure, packet, or recovered graph.

Deterministic fingerprints supply idempotency. If an accepted transition is repeated with the same evidence identity, the existing derived object is returned instead of creating a duplicate. This property is narrower than transactional exactly-once execution: it is verified for the staged candidate and snapshot contracts, not for arbitrary distributed writers.

The table exposes why the versions are cumulative. Orion supplies typed cells and links; Trifid can therefore bind without copying source content into a new authority domain. Lagoon can require replay evidence because Trifid makes replay explicit. Eagle reuses the same evidence-before-mutation rule for procedures. Rosette can distinguish facts from controls because earlier versions assigned them different memory types and link roles.

C. Read and Admission Are Separate

A query does not move directly from the store to the model. Let $\mathcal{R}(q)$ be the deterministic retrieval result, $\mathcal{P}$ the task-level admission policy, and $\mathcal{B}$ the atomic budget operator. The public context contract is

$$F_q, D_q = \mathcal{B}\big(\mathcal{P}(\mathcal{R}(q), o_q), B\big), \tag{10}$$

where o_q contains explicit query controls, B is the context budget, F_q is factual evidence, and D_q is control provenance. D_q can justify why a group is eligible but is not serialized as a factual statement. Global evidence requires opt-in and, in the implemented Orion policy, admission as a resolved winner;

TABLE III
PUBLIC GUARDED-TRANSITION CONTRACT ACROSS V0.00.01–V0.00.06

Transition	Required evidence	Accepted effect	Rejected or repeated effect
Request → runtime result	valid request, loaded generation, bounded context	output and control audit attributed to one runtime	record-level error; no neighboring record reinterpretation
Cells → episode frame	active ordered sources, coherent temporal boundary	episodic anchor and temporal links; sources retained	no frame when sources are missing, inactive, or discontinuous
Completion → replay	complete anchor and ordered source sequence	ordered replay links	no partial replay; repeated replay creates no duplicate links
Replays → schema candidate	complete replay and repeated compatible signature	evidence-only proposal	no semantic write
Candidate → semantic schema	valid fingerprint, sources, and nonempty approval	one semantic cell plus provenance links	state unchanged; repeated approval returns existing cell
Task traces → skill candidate	valid sources and repeated successful strategy; failures retained	evidence-only procedural proposal	insufficient evidence produces no candidate or procedure
Skill candidate → procedure	complete trace mapping and nonempty approval	one procedural cell plus procedure links	state unchanged; duplicate fingerprint is idempotent
Conflict → decision	matching context, opposite outcome, complete review, approval	separate preserve-both control linked to both facts	facts remain; unsupported mode or incomplete review rejected
Facts/controls → packet	valid provenance, eligible group, complete budget unit	factual and control references remain separate	dangling, partial, or procedural-as-fact group rejected
Packet → graph/snapshot	validated packet, typed topology, canonical digest	derived graph or published snapshot	invalid object not published; live state unchanged

stale, retired, losing, and unresolved rows remain available to audit but are excluded from the selected set.

Rosette strengthens this boundary by budgeting groups rather than independent cells. If a policy-approved conflict disposition is `preserve_both`, the two factual sides are one admission unit. Insufficient budget rejects the unit; it cannot silently make a policy choice by retaining only one side. This is a context-integrity property, not an assertion that presenting both claims will make a language model reason correctly.

At the neural boundary, selected factual items are deterministically serialized, truncated only at item boundaries established by the bridge, and encoded as a memory tensor and mask. For frozen model state h and memory path Δh_M, the inherited intervention remains

$$h' = h + \alpha_M \Delta h_M, \tag{11}$$

with a control that sets the memory contribution to zero. The present paper uses this interface but does not repeat the first paper's training and matched baseline claims. It asks whether a governed cell can cross the interface with its identity and control state preserved.

D. What an Audit Record Can Reconstruct

The public audit target is a causal account of the software path, not a full explanation of every neural activation. For one request, the retained record can identify the queried session, cells considered, retrieval tier, selected and excluded identifiers, exclusion reasons, serialized factual items, vector shape and mask, path-control condition, runtime generation, and output artifact. For a derived memory, provenance walks backward through approval, candidate, episode anchor, ordered source cells, and relevant links. For a recovered object, the record adds snapshot schema, digest result, reference validation, topology validation, accepted generation, and skipped generation identifiers.

The trace is append-oriented at the in-memory kernel boundary, but the report does not claim a tamper-proof external audit ledger. A process with direct filesystem or memory access could alter data outside the tested API. The validated statement is that operations performed through the staged interfaces emit inspectable events and that packaged snapshots use canonical content digests and fail-closed loading checks.

VII. SUN V0.00.01: FROM LATENT PATHS TO A RUNNABLE SUBSTRATE

Stages 1–43 established the latent-path and controlled-readout evidence reported in the companion study [7]. The portion expanded here is Stages 44–72, where those experimental components were converted into a bounded inference and artifact-delivery substrate. This is an important architectural transition: a trained Adapter checkpoint is not a system until its input contract, controls, loading behavior, failure states, and outputs are stable enough for later memory versions to depend on them.

A. Task Recovery and Packaging

Stage44 separated local controlled tasks from public-task recovery. Several variants were deliberately rejected. In particular, the hard-negative route did not recover the failing seed and reduced the intended wrong-context effect; the project therefore did not average that failure away. The accepted route reconstructed structured task fields, retained answer-option ownership, and froze a dataset-field cache contract. This history matters because later memory components rely on structured evidence boundaries rather than parsing an opaque concatenated prompt.

Stage45 converted the accepted chain into an Adapter package with a descriptor, checkpoint integrity information, an inference request/response schema, and a single evaluation harness. Stage46 then introduced the runtime backend: requests are validated, text and structured context are encoded, the frozen base produces hidden states, path-specific Adapter computation is applied, and the configured readout returns a response plus a control audit. Stage47 packaged train-only centroids separately from runtime logic and added batch execution. Stage48 defined the production JSONL boundary,

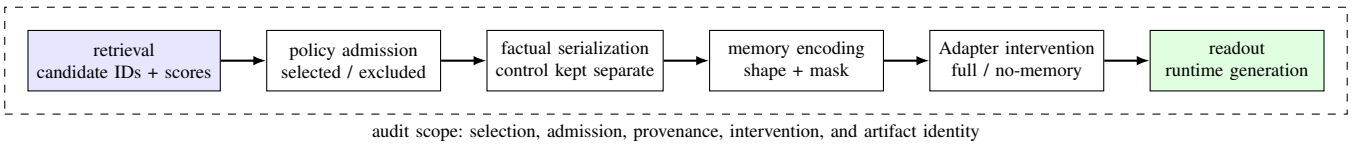

Fig. 3. Public request-level audit chain. It reconstructs the software intervention path but does not claim a complete explanation of neural reasoning.

including per-record validation, bounded context items, explicit ablations, and failure isolation. A malformed record therefore does not silently alter or cancel the interpretation of neighboring records.

B. Service Lifecycle and Concurrency

Stages49–54 moved the runtime into a persistent service and tested the states that are easy to miss in a single-process experiment. The service exposes a health view and serializes runtime ownership. Reload uses a draining state, constructs and validates a replacement before publication, and assigns a runtime version so each response can be attributed to one loaded instance. Real-runtime, concurrent-reload, and long-running stress stages checked that requests did not observe a half-published runtime and that failures remained explicit. These tests validate lifecycle contracts in the reported fixture; they are not a production service-level agreement.

Stages55–59 added deployment packaging, daemon start/stop/status behavior, restart recovery, a diagnostic runbook, and access-control auditing. The paper reports only the public contract: administrative operations are separated from ordinary inference, sensitive values are not copied into audit records, and diagnostics return structured failure reasons. Concrete host, token, and filesystem settings are excluded from the release.

C. Asynchronous Work and Artifact Delivery

Stages60–65 introduced bounded queues, rate limits, asynchronous jobs, persistent job recovery, retention/listing, externalized large results, and batch jobs. The governing principle is indirection: a job record can retain status and a content reference without loading an arbitrarily large result into every service response. Recovery reconstructs explicit job states; it does not treat an interrupted job as silently successful.

Stages66–72 completed the delivery chain. Batch results can be exported, assigned a lifecycle, packaged with a manifest, streamed, requested by byte range, guarded by entity tags and If-Range behavior, and resumed by a client that verifies the final SHA-256 digest. Sun therefore ends not at a model score but at a reproducible delivery boundary that Orion can extend without creating a second service stack.

VIII. Orion v0.00.02: Governed Multi-System Memory

Orion replaces a single undifferentiated context list with four typed memory systems sharing one cell/link/trace substrate. The types are not cosmetic: they define different lifetimes, write origins, retrieval filters, and allowed transitions. Working cells carry time-to-live metadata; episodic cells retain event order and outcomes; semantic cells represent consolidated evidence; procedural cells retain task traces or approved control records.

A. Session Lifecycle and Adapter Bridge

Stage73 established the pure-Python kernel and its core invariants. Stage74 placed one store behind each session and reused the existing Sun service hook: active memory is selected before inference, and working, episodic, and task trace records are written after inference. Context injection remains bounded by the pre-existing request limit. Stage75 extracted serialization into a deterministic bridge that preserves cell identity, ordering, truncation status, vector shape, and mask semantics. The fresh CUDA diagnostic verifies that the same selected cell reaches the service and Adapter path; disabling the memory path reduces the traced deltas to zero while the base model remains frozen.

Stages76–77 introduced replay as an opt-in post-request policy. The policy rejects insufficient evidence without mutation, separates successful and failed outcomes, retains every source episode, and emits identifiers for the derived semantic and procedural cells. Calling this mechanism “consolidation” describes a controlled software transition; it does not mean that the architecture has learned a human-like semantic abstraction.

B. Cross-Session Evidence Governance

Stages78–81 separate session semantics from a global tier. Promotion requires evidence from more than one session for the same task. Global retrieval is disabled by default, and every returned row carries its tier. The Stage81 fixture increases a two-case tier hit rate from 0.5 to 1.0 when global access is enabled; the paper treats this only as a deterministic retrieval contract, not as model accuracy or broad retrieval quality.

Stages82–87 add the governance chain required before global evidence can affect inference. Expired global cells become stale and remain auditable but leave default retrieval. Opposite polarities create a conflict link without overwriting either cell. Retrieval exposes both sides and marks that resolution is required. A confidence-margin policy may record winner and loser metadata, but both remain visible. Finally, a task-level policy admits only an explicitly permitted resolved winner; denied, losing, and unresolved rows remain in the exclusion trace.

Stages88–90 exercise that chain through the real frozen-Qwen service. Denied and allowed requests differ only in their policy controls, and two isolated sessions can reuse the same permitted global winner while retaining separate session traces. The three-condition regression fixture freezes the session-only, global-denied, and global-allowed behavior for later versions.

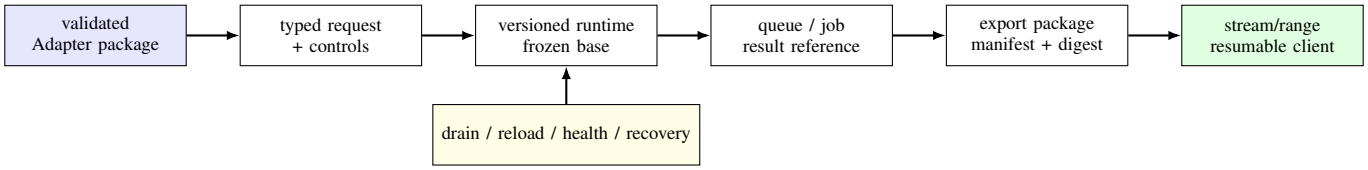

Fig. 4. Sun's public runtime and artifact-delivery contract. Private deployment values are outside the paper boundary.

C. Persistence, Recovery, and Retention

Stages91–99 make the global tier durable without enabling implicit restore. Cells, links, and resolution metadata are saved atomically. Corrupt input returns an isolated empty store and leaves the damaged file unchanged. Save/load is exposed through an explicit lifecycle controller and a protected administrative boundary; restarting a service does not silently populate global memory. Doctor and health views report configuration, file presence, in-memory counts, and lifecycle failures without performing restoration. Retention marks lower-priority cells as retired rather than deleting them, and the state survives save/load. Stage100 closes the release with the explicit boundary that these results demonstrate governance and lifecycle behavior, not long-term task improvement.

IX. Trifid v0.00.03: Episode Binding and Replay

Orion can store episodic cells, but an event may span several observations and cannot be reconstructed safely from lexical similarity alone. Trifid adds an episode frame

$$e = (id, S_o, scene, entities, goal, action, \\ outcome, [t_0, t_1], cues), \tag{12}$$

where S_o is an ordered list of authoritative source-cell identifiers. The frame is an anchor over existing evidence, not a replacement for it.

A. Binding, Disambiguation, and Completion

Stage101 creates a frame and temporal links while retaining source cells. Stage102 requires coherent cue combinations so terms drawn from different episodes cannot be freely mixed into a false match. Stage103 adds episode boundaries and time-window filtering. Stage104 performs pattern completion only when the full ordered source sequence is available; otherwise it returns no partial reconstruction. Reads use the Orion public cell interface so the completion itself produces an audit event.

Stages105–106 make replay explicit and bounded. Replay records ordered links from the anchor to each constituent and is idempotent. The scheduler ranks eligible complete episodes by inspectable fields and applies a budget, but it does not write semantic memory. This separation prevents "selected for replay" from being misreported as "learned knowledge."

B. Candidates, Approval, and Conflict Preservation

Stage107 creates an evidence-only consolidation candidate when multiple complete, replayed episodes share a signature. Stage108 accepts a candidate only with an approval identifier, writes one semantic cell, and records the episode fingerprint for idempotency. Stage109 follows the resulting provenance back from semantic memory to the exact episode anchors; zero lexical match or inconsistent provenance is rejected rather than guessed.

Stages110–113 handle later contradictory episodes. A conflict guard links an opposite outcome to the prior semantic pattern. Review is read-only and requires complete provenance. The implemented decision is deliberately conservative: an approved `preserve_both` record is stored as procedural control evidence, and conflict-aware retrieval exposes the semantic pattern, the contradictory episode, and the decision identifier. It does not rewrite either fact or claim that the conflict has been epistemically solved.

C. Paired Checkpoints and Service Handoff

Stages114–119 persist episode frames alongside the Orion store. A paired checkpoint manifest binds the two snapshots by digest so a store and frame set from different generations cannot be restored together. Recovery walks back to the latest complete generation, retention removes only obsolete valid generations while preserving damaged generations for diagnosis, and audit is read-only. Stage120 freezes these contracts. Stage121 then bridges complete Trifid episode evidence into the existing Orion/Sun service path without changing the frozen model or introducing another runtime endpoint.

X. Lagoon v0.00.04 and Eagle v0.00.05: Controlled Abstraction and Procedure

Trifid supplies complete replay evidence but intentionally stops before broad generalization. Lagoon defines a slower, approval-gated path from episodes to semantic schemas; Eagle applies the same evidence-before-mutation discipline to procedures.

A. Lagoon: Episode-to-Schema Consolidation

Stage122 groups complete replayed episodes only when their structure is compatible and their temporal separation satisfies the configured window. The cluster remains evidence. Stage123 derives a schema candidate with a stable fingerprint, supporting episode identifiers, and aggregate confidence and importance. Stage124 is the only mutation point: a valid approval writes the semantic schema and links every source. Repeated approval returns the existing schema rather than duplicating it.

Stages125–126 make the reverse path first-class. Retrieval validates support episodes, Trifid anchors, and the semantic `consolidated_from` metadata before returning a result. This is stricter than returning a nearest embedding: a schema

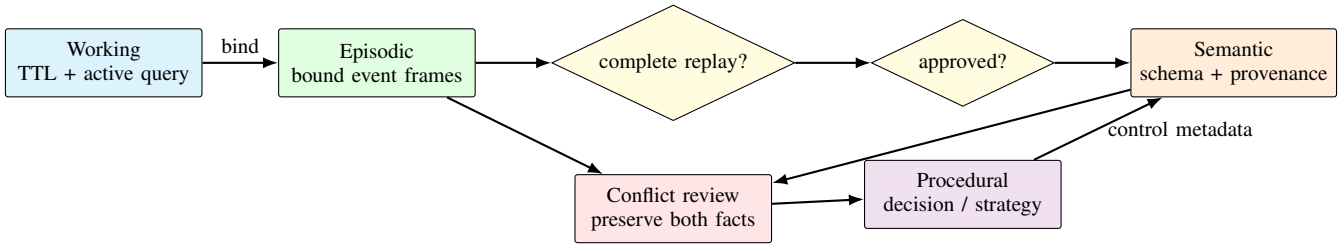

Fig. 5. Memory transitions and approval boundaries. Conflict decisions are control cells; they do not replace factual cells.

with incomplete provenance is not eligible merely because its text is similar.

Stages127–131 handle schema conflicts. A new episode is conflicting only when scene, entities, goal, and action match while outcome differs. The guard adds a conflict link and preserves both objects. Conflict-aware recall can exclude unresolved schemas in strict mode or return them with a `needs_review` state. A review packet is read-only. An approved `preserve_both` decision is stored separately, and decision-aware recall returns both factual sides while excluding the decision cell from factual context. Stage132 verifies the full contract and its artifacts.

B. Eagle: Trace-to-Skill Formation

Eagle starts from normalized task traces rather than from free-form claims. Stage133 canonicalizes task, scene, goal, ordered steps, outcome, sources, and time into deterministic trace identifiers while rejecting missing steps, invalid outcomes, and absent provenance. Stage134 groups traces by task signature, requires repeated successful evidence, keeps failure identifiers, and emits an evidence-only skill candidate. No procedural memory is written at this point.

Stage135 requires explicit approval and writes one procedural skill linked to all supporting traces. Stage136 retrieves a skill only after validating those trace identifiers and procedure links, returning both success and failure counts. Stage137 adds a conflict link when a later failed strategy diverges from the approved procedure; a failure of the same strategy is not misclassified as a competing procedure. The accepted skill and failed trace remain unchanged. Stage138 closes Eagle while rerunning the upstream Lagoon release contract.

The term "skill" therefore has a bounded meaning in this release: an approved, trace-backed sequence suitable for retrieval. It does not denote a learned controller, guaranteed optimal policy, or autonomous tool execution.

XI. Rosette v0.00.06: Atomic Context and Multi-Scale Graph Views

Rosette addresses a failure mode that appears after conflict and procedure records coexist: context assembly can accidentally treat policy records as facts or truncate a preserve-both conflict group to one side. The version is therefore split into a context foundation and a graph projection.

A. Typed Context Packets

Stage139 builds a query-specific packet whose groups distinguish factual cell identifiers from control references. Stable schemas and approved preserve-both cases are represented differently; a decision authorizes a group but is not itself factual content. Stage140 assigns cost and priority to whole groups. Under budget B, a group is admitted only when its complete factual set fits:

$$G' = \{g_j \mid e_j = 1,\ b_j + \sum_{g_i \in G'} b_i \le B\}. \tag{13}$$

No branch admits half of a preserve-both pair.

Stage141 validates the packet against the live Orion store and Trifid binder. It rejects dangling cells, procedural-as-fact errors, malformed conflict groups, and decision links that do not support both sides. Validation is read-only. Stages142–143 add canonical packet snapshots, atomic publication, checksum verification, and live-store validation during recovery. Stage144 is explicitly an intermediate context gate, not the Rosette release.

B. Honeycomb Projection and Traversal

Stage145 projects a validated packet into episode-, schema-, and control-scale nodes. Existing replay, conflict, and task relations become consolidation, conflict, and decision adjacencies. Node identifiers remain tied to source cells, and the graph identifier hashes the canonical packet, query, nodes, and edges. The graph is a derived view; the memory store remains authoritative.

Stage146 provides deterministic bidirectional breadth-first traversal with explicit adjacency, depth, and node budgets. Stage147 compares node types, scales, references, edge vocabulary, and live topology against a rebuild and fails closed on mismatch. Stage148 serializes only a graph that has passed validation. Stage149 combines checksum and live-topology checks during recovery. Stage150 reruns the Lagoon, Eagle, Rosette-context, and Rosette-graph gates and verifies the expected artifact set.

Rosette's contribution is consequently structural, not geometric learning. The controlled graph demonstrates typed construction, filtering, traversal, validation, and recovery. It does not demonstrate learned edge weights, large-scale graph retrieval, or superior graph reasoning.

XII. End-to-End Evidence Walkthrough

Stage tables identify responsibilities but do not by themselves show how an object moves through the system. This

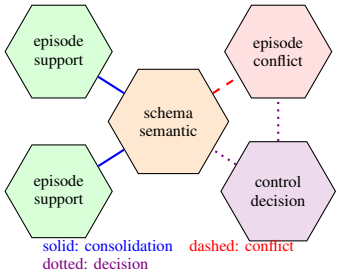

Fig. 6. Rosette controlled graph fixture: five nodes, three scales, and three typed adjacencies.

section follows the controlled public fixture used by the later release runners. Individual stage runners rebuild their own deterministic fixture, so local identifiers are not presented as one continuous production database. The continuity being tested is the contract: each downstream object must be explainable by the same classes of upstream evidence and guards.

A. From Observations to a Semantic Proposal

The fixture begins with time-indexed observations of a task in a laboratory. Trifid binds the observations into episode frames containing scene, entity, goal, action, outcome, ordered source identifiers, and temporal boundaries. The sources remain in Orion; the new anchor is an index over them. Cue and time filtering must identify one coherent frame. Pattern completion then requests the anchor and every source through the public read interface. If one source is absent or inactive, completion returns no partial sequence.

Replay adds ordered anchor-to-source links. It does not create semantic memory. In the release fixture, two compatible replayed episodes fall within the consolidation interval, while a later otherwise compatible episode is separated into another temporal region. Lagoon therefore emits one cluster containing the two nearby episodes and explicitly reports zero semantic writes at that stage. The next stage creates a candidate that retains its support episode identifiers, aggregate confidence and importance, structural signature, and deterministic fingerprint. Only the approved consolidator creates the semantic cell; an empty approval is rejected and a repeated fingerprint returns the existing cell.

This sequence separates four questions that would otherwise be conflated: whether observations exist, whether they form a coherent episode, whether several episodes support a proposal, and whether policy permits that proposal to become semantic memory. The release gate exercises a bounded instance of each question; it does not establish that the hand-coded signature discovers all useful abstractions.

B. Conflict Without Factual Overwrite

A later episode in the fixture has the same scene, entities, goal, and action but an opposite outcome. The Lagoon guard adds a typed conflict link from the new episode anchor to the semantic schema. Repeating detection does not add a second equivalent link, and an episode from a different context is not marked as this conflict. Before approval, conflict-aware recall can label the schema as requiring review and strict mode can exclude it.

The review step checks the schema, its support episodes, their anchors, the opposing episode, and the conflict link without mutating any of them. The implemented decision mode is deliberately limited to `preserve_both`. Approval creates a procedural control cell and links it to both factual sides. It does not alter either outcome or remove the conflict edge. Decision-aware recall consequently returns two factual cell identifiers and one control reference. This is a policy for preserving uncertainty, not an automatic fact resolver.

C. A Parallel Procedure Lineage

Eagle exercises a separate path over task traces. The controlled fixture contains repeated successful executions of one ordered strategy and a failed trace. Normalization rejects traces with empty steps, unsupported outcomes, or missing sources and derives stable identities from canonical public fields. Candidate construction chooses the most supported successful step sequence while retaining the complete trace set and the failed-trace identifiers.

The candidate remains non-authoritative until approval. The approved skill is a procedural cell linked to each supporting trace. Retrieval revalidates the trace identifiers, procedure links, source cells, strategy steps, and outcome counts before returning the procedure. A later failed trace with the same strategy is not treated as a competing strategy; a failed trace with a different step sequence receives a conflict link. Neither case rewrites the approved procedure. Thus “skill” means a retrievable, approved, trace-supported sequence in this release, not a learned action policy.

D. From Policy-Aware Recall to a Graph View

Rosette assembles the approved conflict case into a typed packet. The fresh Stage139 artifact contains one eligible preserve-both group with two factual cells, one control cell, two supporting episodes, and one conflicting episode. The exact identifiers are retained in the evidence package but are not needed to understand the public contract. A stable group is prioritized by the budget router, and a preserve-both group is either admitted with both facts or omitted. The associated control reference follows the group but remains in a separate field.

The graph builder expands the validated packet into five nodes: two supporting episode nodes, one conflicting episode node, one semantic-schema node, and one control-decision node. Five edges cover consolidation, conflict, and decision adjacencies. A full traversal from the schema reaches all five nodes at depth at most one in this fixture; adjacency-filtered traversals isolate the expected support or conflict neighborhood, and a node budget truncates traversal deterministically. These observations verify graph construction and traversal semantics for the fixture. They are not a graph-reasoning benchmark.

TABLE IV
CONTROLLED PUBLIC WALKTHROUGH: OBSERVABLE STATE AND NEGATIVE CONTROL

Boundary	Positive observation	Negative control	Supported interpretation
Episode binding	anchor references ordered active source cells	mixed cues, time gaps, and missing sources rejected	coherent fixture-level episode identity
Replay	ordered links cover every completed source	no partial or duplicate replay; no semantic write	explicit replay evidence
Lagoon candidate	two nearby compatible episodes form one proposal	distant episode split; insufficient support writes nothing	evidence-only schema proposal
Approved schema	one semantic cell retains support and fingerprint	empty approval rejected; repeated fingerprint idempotent	guarded semantic mutation
Conflict decision	both factual sides and separate control survive	different context ignored; incomplete review rejected	preserve-both policy, not truth selection
Eagle skill	approved procedure links supporting success/failure traces	insufficient success and missing provenance rejected	trace-backed retrievable procedure
Rosette packet/graph	facts and controls remain typed; five-node graph covers three adjacencies	partial group, dangling reference, and invalid topology rejected	bounded derived context and graph view
Recovery	valid generation or snapshot published after all checks	corruption, hash mismatch, and topology mismatch fail closed	tested publication integrity

E. Artifact Publication and Recovery

Packet and graph snapshots are accepted only after canonical serialization, digest verification, schema validation, and comparison with live cells and links. The recovery fixtures independently inject malformed JSON, changed canonical payloads, dangling cell references, and topology disagreement. A valid object is returned only when every check passes. Invalid recovery returns a structured failure and does not modify the live store or publish a partially loaded view.

Trifid applies the same principle at a different boundary. Its checkpoint manifest binds an Orion store snapshot and a frame snapshot. If the newest generation is damaged, recovery skips the entire pair and loads the newest earlier valid generation; it never combines a newer store with older frames. Retention removes obsolete valid generations only after retaining a current valid one, while damaged generations remain available for diagnosis. These mechanisms explain the report's use of *fail closed*: it refers to object publication under the tested corruption classes, not immunity to arbitrary storage or process failures.

F. What the Walkthrough Establishes

The walkthrough demonstrates a complete, inspectable chain from source cells to episode structure, replay evidence, approval-gated semantic and procedural objects, conflict-preserving context, graph projection, and validated recovery. At every mutating boundary, a corresponding negative case verifies that missing evidence or authorization leaves authoritative state unchanged. At every derived-view boundary, validation can reject the view without rewriting the store.

It does not demonstrate open-domain retrieval quality, autonomous consolidation, correct resolution of real contradictory claims, improved language-model accuracy, or production-scale persistence. Those require the held-out, multi-instance baseline study described as future evaluation rather than inferred from release tests.

XIII. FAILURE-DRIVEN EVOLUTION AND NON-REGRESSION

The version line was not produced by retaining only successful final runs. Failures were used to narrow the next admissible design. Sun's early public task migration showed that local path separability did not imply external task transfer. Stage44's rejected hard-negative route showed that lexical negative construction could weaken rather than strengthen wrong-context dependence. The accepted repair moved task boundaries into structured fields and made the cache schema auditable.

The memory versions apply the same discipline to state. Insufficient replay must reject without mutation; a candidate must not become knowledge without approval; an opposite outcome must not overwrite its predecessor; a decision must not masquerade as a fact; a tight context budget must not choose one side of a preserved conflict; and a corrupt or topologically inconsistent snapshot must not publish a partial object. Each constraint is represented by an executable negative case rather than only by prose.

Non-regression is hierarchical. A version release reruns the contracts on which it directly depends, while the final Stage150 gate nests Lagoon, Eagle, Rosette context, and Rosette graph checks. The fresh paper reproduction adds separate Orion and Trifid test groups and compares local and WSL source manifests. This does not prove correctness for every possible state, but it provides a traceable explanation for why a later feature was accepted and which earlier behavior it was required to preserve.

XIV. COMPLETE PUBLIC STAGE MAP

Tables VI–XI enumerate the public responsibility of every stage from the post-companion Sun work through Rosette. The map intentionally omits private configuration, payload content, and parameter artifacts. "Gate" denotes the behavior that allowed the next stage to depend on the result; it is not a claim of general task performance.

A. How the Stage Sequence Should Be Read

The 107 integer indices from Stage44 through Stage150 are not 107 independent scientific hypotheses. They form a failure-isolation protocol with four recurring stage roles. A *construction stage* introduces one public object or operation, such as an Adapter package, episode frame, schema candidate, task trace, context packet, or graph. A *guard stage* defines the cases in

TABLE V
FAILURE-TO-CONSTRAINT LINEAGE ACROSS THE VERSION SERIES

Observed risk or failure	Rejected interpretation	Constraint introduced	Version
Local path success without public transfer	latent separation solves external tasks	structured task boundaries and explicit negative reporting	Sun
Hard negatives weaken wrong-context dependence	more lexical negatives guarantee grounding	dataset-field evidence contract and fail-fast per-seed gates	Sun
Too few episodes for replay	any event can become semantic memory	reject without mutation; require complete replay evidence	Orion/Trifid
Opposite outcomes under one context	newest or highest score is automatically true	preserve both cells and add an explicit conflict relation	Orion/Lagoon
Decision record appears beside facts	policy metadata is factual evidence	separate fact and control references; validate before context assembly	Lagoon/Rosette
Tight context budget splits a conflict group	truncation is a neutral operation	admit or reject the complete factual group atomically	Rosette
Corrupt or mismatched snapshots	partial recovery is better than failure	checksum, reference, generation, and live-topology fail-closed gates	Orion–Rosette

which that object must be rejected, excluded, or preserved. A *lifecycle stage* tests persistence, restart, retention, delivery, or recovery. A *release stage* re-executes its dependencies and verifies their artifacts rather than trusting an earlier status flag.

This organization is important for interpreting apparent repetition. For example, Trifid uses separate stages for snapshot creation, snapshot recovery, paired checkpoints, generation fallback, retention, and read-only audit. A single "persistence passed" stage would hide materially different failure surfaces: a syntactically valid frame file can still refer to absent cells; two individually valid files can belong to different generations; and cleanup can destroy the only valid fallback. The narrow stages expose those cases as separate contracts.

The same pattern appears in Rosette. Packet construction does not imply that the packet is valid against a live store, and packet validation does not imply that a serialized copy can be recovered after the store changes. Graph construction, traversal, topology validation, snapshot publication, and recovery are therefore distinct. Stage150 accepts the release only after the context foundation, graph chain, Lagoon dependency, Eagle dependency, stage set, and artifact set all pass.

B. Dependency and Non-Regression Closure

The dependency relation is directional. Later stages may read or derive from earlier objects, but a derived view may not retroactively redefine its source. Let D_j be the direct dependencies of stage j and G_j its local gate. The accepted status is

$$A_j = G_j \wedge \bigwedge_{k \in D_j} A_k. \tag{14}$$

In practice, not every ordinary stage recursively runs the complete history; release runners provide the closure at version boundaries. The evidence package therefore records both local functional assertions and the nested release artifact tree. A missing required artifact fails release even when a summary file says the local stage passed.

Sun differs from the later memory versions because its post-companion stages include a larger operational surface. Stage44 records the accepted and rejected task-recovery branches; Stages45–54 stabilize package, request, runtime, service, and reload behavior; Stages55–72 extend operations and artifact delivery. The present reproduction maps those historical contracts but does not label them as one newly rerun end-to-end service benchmark. The fresh Stage75 diagnostic exercises the inherited neural bridge needed by Orion, while the fresh Orion, Trifid, and Lagoon–Eagle–Rosette regression groups cover the current memory implementation.

The tables use short phrases because their purpose is traceability, not to replace the mechanism sections. A stage row answers three questions: what public responsibility was added, what failure boundary was tested, and what later component was allowed to depend on it. Quantitative fixture results and their evidential limits are reported separately in Sections on evaluation and results.

XV. IMPLEMENTATION AND SAFETY BOUNDARIES

The kernel and staged policies are implemented as pure Python modules. The kernel depends on neither a vector database nor PyTorch. Model dependencies enter only at the optional adapter bridge. This boundary permits memory lifecycle tests to run without loading Qwen3, while the CUDA diagnostic tests the integration contract separately.

A. Write, Read, and Audit Path

A write validates the memory type and bounded numeric fields, allocates a system-prefixed identifier, stores the cell, and appends a write event. A read first expires due working cells, applies system/source/context filters, scores active candidates with (3), applies a stable ordering, and appends a read event containing returned identifiers. Link creation validates both endpoints and the fixed type vocabulary before appending a link event. Conflict handling invokes the same link operation and emits a conflict trace; there is no overwrite branch.

B. Approval-Gated Transitions

Candidate builders are read-only. A consolidator rejects missing approvals, missing source cells, incomplete replay, and mismatched fingerprints. On success it writes the derived cell and provenance links. Repeating the same candidate returns the existing cell. Thus mutation is represented as

$$(C, L, A) \xrightarrow[approval]{candidate} (C \cup \{c^*\}, L \cup L^*, A \| A^*), \tag{15}$$

while an invalid candidate leaves the state unchanged.

TABLE VI
SUN STAGE MAP I: TASK RECOVERY, PACKAGING, RUNTIME, AND SERVICE LIFECYCLE

Stage	Public responsibility	Gate or boundary
44A	consolidate local multiclass tasks under one runner	retain controlled path and context-ownership checks
44B audit	reconstruct public-task source fields and deterministic splits	reject leakage, invalid option ownership, and incompatible checkpoints
44B.0	initial external recovery run	negative result retained; no public-transfer claim
44B.1	task-specific readout and BoolQ boundary repair	task boundary and frozen-base checks
44B.1-M	repeat external recovery over three seeds	every seed must pass; averaging cannot hide failure
44B.2	RTE/BoolQ focused repair	failing seeds remain explicit
44B.3	cross-example hard-negative route	rejected when wrong-context dependence weakened
44B.4	structured semantic wrong-context route	three-seed structured-context gate
44B.5	persist structured external cache schema	remove dependence on runtime tail parsing
44B.6	rebuild cache from dataset fields	field-level manifest and fail-closed missing-cache behavior
44C	unify accepted local and external runners	both branches and cache contract required
45	freeze Adapter package, request schema, and evaluation harness	package integrity and upstream-artifact validation
46	expose Qwen3 frozen runtime API	controls, readout, finite outputs, and frozen weights
47	package train-only fixed centroids and batch inference	centroid provenance and per-request outputs
48	define bounded JSONL inference with ablations	record-level validation, control trace, and error isolation
49	persistent inference service	health and request lifecycle use one runtime owner
50	service stability stress	repeated requests preserve status and control semantics
51	draining, versioned runtime management and reload	replacement validates before atomic publication
52	real-runtime reload stress	no half-loaded runtime becomes visible
53	concurrent request/reload stress	each response identifies one coherent runtime version
54	long-running service stress	errors and resource observations remain explicit

TABLE VII
SUN STAGE MAP II: OPERATIONS, ASYNCHRONOUS JOBS, AND ARTIFACT DELIVERY

Stage	Public responsibility	Gate or boundary
55	deployment package and client-facing service wrapper	deployment artifact validates before use
56	daemon start/stop/status operations	idempotent lifecycle and structured status
57	daemon recovery	stale state and failed startup remain diagnosable
58	service doctor and runbook checks	report-only diagnosis separated from mutation
59	access-control and audit boundary	administrative operations protected; sensitive values redacted
60	bounded queue and rate limiting	overload is rejected explicitly rather than silently dropped
61	asynchronous job management	stable job identifiers and terminal-state reporting
62	persistent job recovery	interrupted jobs recover to explicit states
63	job retention and listing	bounded retained history with deterministic listing
64	external result store	large payload referenced rather than copied into job metadata
65	batch jobs	one batch preserves per-record result/error identity
66	batch export	export is generated from completed result references
67	export lifecycle	export status and cleanup are explicit
68	export package delivery	package includes manifest and content integrity data
69	streaming delivery	bounded streaming without loading the whole package into a response
70	HTTP byte-range delivery	valid ranges served; invalid ranges rejected
71	ETag/If-Range semantics	changed entities prevent unsafe continuation
72	resumable download client	resumed bytes must pass final SHA-256 verification

TABLE VIII
ORION STAGE MAP: FOUR-SYSTEM MEMORY, GLOBAL GOVERNANCE, AND DURABILITY

Stage	Public responsibility	Gate or boundary
73	typed working/episodic/semantic/procedural kernel	TTL, timeline, links, traces, and conflict preservation
74	per-session memory service integration	session isolation and bounded pre-read/post-write hooks
75	cell-to-Adapter context bridge	deterministic provenance, finite vectors, zero no-memory delta
76	replay/consolidation policy	insufficient evidence rejects without mutation
77	request-level replay lifecycle	disabled by default; accepted/rejected outcome traced
78	cross-session semantic promotion	require evidence from multiple distinct sessions
79	session/global retrieval tiers	global disabled by default and tier annotated
80	request-level global opt-in	no implicit global effect on session inference
81	deterministic retrieval-quality fixture	tier hit contract only; no model-accuracy interpretation
82	global staleness and polarity conflicts	stale excluded; conflicting cells retained
83	conflict-aware retrieval	both peers returned with resolution-required status
84	confidence-margin resolution metadata	unresolved margins preserved; no deletion
85	resolution-aware trace	winner, loser, peer, and unresolved state visible
86	task-level global admission policy	only permitted resolved winners selected
87	service injection of admission decision	selected/excluded provenance and tier-prefixed identity
88	real CUDA denied/allowed diagnostic	policy controls path; Qwen remains frozen
89	multi-session global evidence reuse	shared winner with isolated session traces
90	three-condition governance regression	session-only, denied, and allowed behavior frozen
91	atomic global-store persistence	cells, links, and resolution metadata restore
92	corruption recovery contract	damaged input preserved; isolated empty store returned
93	explicit save/load lifecycle controller	no implicit service-start load
94	protected lifecycle API	fixed configured path; unauthorized access rejected
95	restart recovery and lifecycle metrics	new service empty until explicit load
96	doctor/restore helper	report by default; restoration requires explicit action
97	health and administrative observability	status views do not trigger restore
98	active-capacity retention	low-priority cells retired, not deleted
99	retention/persistence regression	retired/stale states survive save/load and remain visible
100	Orion release gate	governance and lifecycle accepted; accuracy claim explicitly excluded

TABLE IX
TRIFID STAGE MAP: EPISODE STRUCTURE, REPLAY, CONFLICT, AND CHECKPOINTS

Stage	Public responsibility	Gate or boundary
101	bind ordered source cells into an episode frame	sources retained; frame and temporal links auditable
102	disambiguate episodes by coherent cue sets	mixed cues from different episodes do not spuriously match
103	temporal index and episode boundaries	discontinuous sources and out-of-window episodes excluded
104	pattern completion	return full ordered evidence or no completion
105	idempotent episode replay	ordered replay links; no automatic consolidation
106	budgeted replay scheduling	deterministic eligible ranking and repeated-run idempotency
107	consolidation candidate generation	complete replay and minimum signature support; zero semantic writes
108	approved semantic consolidation	approval required; fingerprint prevents duplicates
109	semantic-to-episode provenance retrieval	exact anchors and source order; inconsistent provenance rejected
110	semantic/episode conflict guard	opposite outcome linked; both sides retained
111	read-only conflict review	complete provenance required; no state mutation
112	approved preserve-both decision	control cell linked to both sides; conflict remains
113	conflict-aware retrieval	fact, conflicting episode, and decision provenance surfaced
114	episode-frame snapshot	schema/reference checks and monotonic episode identity
115	snapshot recovery	corrupt or dangling snapshots fail closed without rewrite
116	paired Orion/Trifid checkpoint	manifest binds store and frame snapshots by SHA-256
117	generation fallback	skip damaged generation; never publish a half pair
118	checkpoint retention	keep current valid generation and damaged evidence for diagnosis
119	read-only checkpoint audit	report integrity and store/frame counts without modifying files
120	Trifid release gate	binding, replay, approval, conflict, and checkpoint contracts pass
121	bridge complete episode evidence into Stage74 service	reuse existing runtime boundary; no model training

TABLE X
LAGOON AND EAGLE STAGE MAP: SEMANTIC SCHEMAS AND PROCEDURAL SKILLS

Stage	Public responsibility	Gate or boundary
122	cluster compatible replayed episodes	temporal split, complete evidence, zero semantic writes
123	derive evidence-only schema candidates	support identifiers and stable fingerprint retained
124	approved schema consolidation	one idempotent semantic write with provenance
125	schema retrieval	read derived schema without extra semantic mutation
126	strict provenance query	validate schema, support episodes, anchors, and conflict state
127	schema conflict guard	same context/opposite outcome only; add link, preserve facts
128	conflict-aware schema recall	stable context admitted; unresolved optionally excluded
129	read-only conflict review packet	reject non-Lagoon or incomplete provenance
130	approved preserve-both decision	idempotent control record linked to both facts
131	decision-aware recall	return both facts; control cell excluded from factual context
132	Lagoon release gate	ten stages and required cell/link/trace artifacts verified
133	normalize task traces	deterministic IDs; sources/time/outcome retained; invalid input rejected
134	build skill candidates	repeated success required; failure traces retained; no procedure write
135	approved skill consolidation	one skill linked to every supporting trace
136	skill retrieval with provenance	trace IDs, procedure links, and outcome counts validated
137	skill conflict guard	divergent failed strategy linked; skill and failure unchanged
138	Eagle release gate	Eagle contracts pass while Lagoon remains green

TABLE XI
ROSETTE STAGE MAP: CONTEXT PACKETS AND HONEYCOMB GRAPHS

Stage	Public responsibility	Gate or boundary
139	build typed context packet	factual cells and control references remain separate
140	atomic context-budget routing	preserve-both group admitted or rejected as one unit
141	live-store packet validation	procedural-as-fact, dangling, and partial groups rejected
142	canonical packet snapshot	schema version, digest, and atomic file publication
143	packet recovery	checksum and live-store validation; invalid object not published
144	context-foundation release gate	Lagoon/Eagle remain green; not yet Rosette completion
145	build multi-scale honeycomb graph	episode/schema/control nodes and three typed adjacencies
146	deterministic graph traversal	adjacency, depth, and node budgets enforced
147	live-topology graph validator	type, scale, reference, vocabulary, and rebuild comparison
148	validated graph snapshot	invalid graph cannot be serialized as accepted state
149	graph recovery	checksum plus live-topology agreement required
150	final release gate	complete v0.00.04–v0.00.06 chain and required artifacts verified

TABLE XII
PUBLIC ARTIFACT CONTRACT ACROSS THE SIX VERSIONS

Version	Public artifact class	Integrity or provenance field	Authority boundary
Sun	Adapter/runtime descriptors, job and export manifests	package/checkpoint identity, runtime version, content digest	frozen runtime and validated package
Orion	cells, links, trace events, global-store snapshot	source, time, tier, policy decision, lifecycle status	typed store; global use is opt-in
Trifid	episode frames, replay links, paired checkpoints	ordered source IDs, episode fingerprint, generation digest	Orion cells remain authoritative
Lagoon	clusters, schema candidates, approved schemas, conflict decisions	support episodes, approval ID, schema/conflict fingerprint	candidate is evidence; approved schema is derived fact
Eagle	normalized traces, skill candidates, approved procedures	source traces, outcomes, approval ID, procedure links	candidate is evidence; approved skill is procedural memory
Rosette	context packets, honeycomb graphs, snapshot manifests	fact/control groups, node/edge source IDs, canonical digest	packet/graph is a validated query view

TABLE XIII
VALIDATION LAYERS AND THE STRONGEST PERMITTED INTERPRETATION

Evidence type	What it directly checks	Strongest permitted conclusion
Unit/functional assertion	one typed transition, filter, state, or artifact condition	implemented contract executes for the fixture
Negative/failure injection	rejection, idempotency, preservation, or fail-closed branch	specified unsafe transition is blocked in the fixture
Nested release gate	required prior runners, stage set, and artifact presence	reported version chain is internally reproducible
Fresh Stage75 CUDA diagnostic	selected cell, vector/mask, Adapter delta, no-memory control, frozen weights	memory item reaches a controllable frozen-model path
Inherited Stage42 multi-seed result	fixed-centroid behavior over three controlled seeds	inherited local path robustness only
Not included	held-out long-horizon tasks, matched RAG, scale/load, cross-model tests	no predictive, production-scale, or universal-memory claim

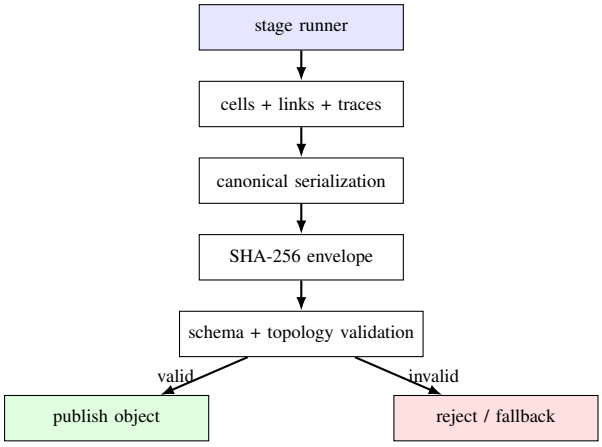

Fig. 7. Fail-closed artifact path. A summary alone cannot publish a recovered store, packet, checkpoint, or graph.

C. Atomic Context Budget

Let groups $g_j = (F_j, D_j, b_j, e_j)$ contain factual cells F_j, control cells D_j, token cost b_j, and eligibility e_j. The context assembler sorts eligible groups deterministically and admits a group only if $\sum b_j \leq B$. It never truncates F_j. In particular, a preserve-both pair cannot be reduced to one side by a tight budget. This guards against a budgeting operation silently becoming a conflict-resolution policy.

D. Graph Construction and Traversal

Rosette builds a query-specific graph only from a validated packet. Node IDs derive from cell IDs; graph IDs are SHA-256 digests over canonical packet, query, node, and edge records. Traversal is deterministic breadth-first search over bidirectional views of typed edges, with adjacency, depth, and node-budget filters. The graph is a view: memory cells and links remain authoritative.

E. Persistence and Recovery

Store, packet, checkpoint, and graph snapshots use canonical JSON and integrity hashes. Recovery verifies schema, digest, references, and live topology before publishing a loaded object. Failed validation does not return a partial object. Retention removes obsolete generations only after a current valid generation exists. Fig. 7 shows this fail-closed path.

XVI. EVALUATION PROTOCOL

Evaluation has three distinct layers. Layer 1 reuses the controlled Sun model-path benchmark to establish the inherited latent intervention surface. Layer 2 runs staged functional tests over the software memory architecture. Layer 3 runs a real frozen-Qwen CUDA diagnostic to verify the cell-to-vector and vector-to-adapter handoff. Results from one layer are not interpreted as results for another.

A. Environment and Reproduction

The fresh reproduction was executed on a WSL Ubuntu host using Python 3.12.13 and an NVIDIA GeForce RTX 5060 Ti with 16,311 MiB reported memory. Before execution, SHA-256 manifests for all staged analysis and test files were compared between the paper workspace and WSL; the manifests were identical. Three non-overlapping test groups were then run: Orion Stages 73–99, Trifid Stages 101–121, and Lagoon–Eagle–Rosette Stages 122–150. Stage75 and Stage150 runners were executed again into fresh output directories, which were copied into the paper evidence package.

The Sun Stage44–72 map is based on the version's implementation modules, stage reviews, and existing release artifacts. This paper did not rerun the entire historical service/deployment suite as one fresh command. The fresh Stage75 service diagnostic exercises the inherited request, frozen-runtime, Adapter, and control path needed by Orion, but it is not a substitute for a new full Sun operational benchmark. The evidence ledger marks this distinction so the stage history is not presented as a fresh performance result.

The previously published Sun multi-seed result is reused as inherited evidence rather than silently relabeled as a new run. Its artifact contains seeds 202, 303, and 404, train-only fixed-centroid evaluation, frozen-weight checks, and per-seed failure fields. The current paper verifies and packages that artifact; the fresh experiments concern the memory versions and integration path.

B. Functional Gates

Each stage defines assertions over both behavior and artifact presence. Release gates call prior stage runners rather than trusting a status file. The final Stage150 gate recursively executes Lagoon, Eagle, Rosette context, and Rosette graph stages, then checks seven release conditions. This design

TABLE XIV
FRESH WSL REPRODUCTION EVIDENCE

Test group	Tests	Time (s)	Result
Orion 73–99	40	13.67	pass
Trifid 101–121	20	8.71	pass
Lagoon–Rosette 122–150	29	9.56	pass
Stage75 CUDA runner	7 gates	–	pass
Stage150 artifact runner	7 gates	–	pass

detects missing files even when an earlier summary says "pass."

C. Controlled Fixtures

Fixtures are deliberately small so failures are attributable. Lagoon uses six episode cells in its cluster fixture. Eagle normalizes four traces (three successful and one failed) and produces one candidate. The Rosette graph fixture contains five nodes and five edges across three scales and three adjacency types. These counts are structural coverage statistics, not task benchmarks.

D. Failure Injection

The suites exercise expired working cells, incomplete replay, missing approval, duplicate consolidation, contradictory outcomes, missing graph references, mutated canonical payloads, and corrupted latest checkpoints. Expected behavior is exclusion, rejection, idempotent reuse, preserve-both linking, or fallback to a previously valid generation. Table XV reports these contracts.

XVII. EVIDENCE AND CLAIM TRACEABILITY

The release package distinguishes historical design evidence, fresh regression evidence, controlled integration evidence, and structural fixture evidence. This distinction is necessary because a passing unit test, a CUDA path diagnostic, and a predictive benchmark answer different questions. The evidence ledger records the source manifest, exact command logs, regenerated Stage75 and Stage150 artifacts, and the inherited Stage42 artifact separately. The manuscript uses a claim only at the strongest level supported by its artifact.

For a claim q, define its evidence record as

$$\mathcal{E}(q) = (h_s, h_d, h_r, \rho, \gamma, \nu), \tag{16}$$

where h_s is the source-manifest hash, h_d is the data or artifact hash, h_r is the runner-output hash, ρ is the reproduction mode, γ is the gate result, and ν is the scope limitation. A public claim is admissible only when all available fields are retained in the evidence bundle. For example, the Stage75 claim is a fresh CUDA integration result with a frozen-weight audit; it is not promoted to an accuracy claim because its fixture contains one designed request.

The fresh WSL reproduction used non-overlapping test commands for Orion, Trifid, and Lagoon–Eagle–Rosette. Source manifests were compared before execution, then the Stage75 adapter diagnostic and Stage150 release runner were regenerated in empty output locations. The resulting package contains the runner outputs rather than a manually transcribed pass/fail status. The Stage150 artifact chain also checks that required cells, links, traces, packets, and graph snapshots exist; a stage summary is insufficient on its own.

This protocol intentionally leaves a gap between contract correctness and predictive performance. No open-domain dataset, learned retriever, vector database, long-horizon user study, or matched RAG benchmark is included in this release. Such results require independently held-out tasks, fairness controls, and multi-instance reporting. The evidence package therefore makes the architecture inspectable now while preserving a clear boundary for later performance studies.

XVIII. RESULTS

A. Fresh Release Reproduction

All three current test groups passed: Orion reported 40 tests in 13.67 s, Trifid reported 20 tests in 8.71 s, and Lagoon–Eagle–Rosette reported 29 tests in 9.56 s. These are current test-discovery counts. Historical release notes reported 34 Orion tests, 26 tests for the Trifid release command (which also included selected Orion tests), and 29 final-chain tests. We retain both sets in the ledger and use the fresh counts for the present reproduction.

The newly generated Stage150 tree contained 153 files. Its seven top-level gates were true: Lagoon release, Eagle release, Rosette context release, all honeycomb stages, complete stage set, artifact presence, and roadmap-contract coverage. The summary SHA-256 is included in the evidence ledger.

Across the nested release artifacts, Lagoon contributes 10 functional stages and 59 asserted gates, Eagle contributes five stages and 30 gates, Rosette context contributes five stages and 32 gates, and Rosette graph contributes five stages and 32 gates. All 153 functional assertions passed. The graph fixture contains two support nodes, one semantic node, one contradictory episode, and one decision node. Full traversal visits all five; adjacency filters isolate semantic–support and semantic–conflict neighborhoods.

B. Frozen-Model Integration

The fresh Stage75 CUDA diagnostic encoded one semantic cell as a finite $[1, 1, 1024]$ tensor with a $[1, 1]$ true mask. The full control produced memory-delta norms 7.7929 and 4.9193 at the traced adapter locations. The same request under `no_memory_path` produced 0.0 and 0.0. Both controls reported zero Qwen trainable parameters, zero Qwen gradients, unchanged weights, and successful provenance retrieval. This establishes an executable intervention boundary. Because the diagnostic contains one designed request, it does not measure predictive accuracy.

C. Inherited Sun Path Evidence

The packaged Sun Stage42 artifact reports three of three seeds passing a controlled five-candidate fixed-centroid gate. Mean accuracy increased from 0.2049 ± 0.0028 before alignment to 1.0000 ± 0.0000 after alignment; wrong-context drop was 0.8000 ± 0.0000 and hidden-norm ratio was $1.1270 \pm$

TABLE XV
REPRESENTATIVE FAILURE-INJECTION CONTRACTS

Injected condition	Required response	Preserved evidence	Covered layer
working TTL elapsed	exclude from active query	expired cell + expire trace	Orion
incomplete replay	reject candidate	episode and partial links	Trifid
missing approval ID	reject derived write	all source cells	Trifid/Lagoon/Eagle
repeat fingerprint	return existing cell	original provenance	consolidation
opposite outcome	add conflict relation	both factual cells	Lagoon
tight context budget	drop whole fact group	preserve-both pair	Rosette context
mutated snapshot payload	fail closed	prior valid generation	recovery
missing graph cell	reject graph/recovery	authoritative store	Rosette graph

TABLE XVI
CLAIM-TO-EVIDENCE MAP FOR THIS PREPRINT. "FRESH" DENOTES A NEW WSL OR CUDA REPRODUCTION PACKAGED WITH THIS RELEASE; "INHERITED" DENOTES A SEPARATELY IDENTIFIED RESULT FROM THE FIRST STUDY.

Claim	Direct artifact	Evidence status	Explicit scope limit
Version contracts execute	WSL regression logs: 40, 20, and 29 tests	Fresh	functional tests, not workload performance
Cell reaches frozen adapter path	Stage75 summary: shape, deltas, frozen audit	Fresh CUDA	one designed request, not task accuracy
Rosette release artifacts are complete	Stage150 summary and 153-file tree	Fresh	fixture and release coverage only
Graph traversal covers typed structure	Stage145 controlled graph summary	Fresh	five nodes, not a graph-reasoning benchmark
Latent-path robustness exists	Stage42 three-seed summary	Inherited	local five-candidate controlled benchmark

TABLE XVII
RELEASE-CONTRACT COVERAGE AND PUBLIC EVIDENCE ARTIFACTS.

Scope	Fresh command result	Primary contract	Regenerated artifact	Interpretation
Orion 73–99	40 pass / 13.67 s	typed store, policy, persistence	regression log	contract coverage
Trifid 101–121	20 pass / 8.71 s	binding, replay, checkpoints	regression log	contract coverage
Lagoon–Rosette 122–150	29 pass / 9.56 s	consolidation, skills, graph	Stage150 tree	contract coverage
Stage75 CUDA	7/7 gates	cell-to-adapter intervention	diagnostic summary	integration boundary
Stage150 release	7/7 gates; 153 files	nested release completeness	summary + artifacts	artifact completeness

TABLE XVIII
CONTROLLED FIXTURE STATISTICS

Artifact	Cells	Links	Traces
Lagoon cluster	6	6	16
Lagoon conflict	9	11	28
Lagoon decision	8	11	28
Eagle skill	5	4	10
Eagle conflict	7	5	13
Rosette packet	8	11	40
Rosette graph	8	11	42

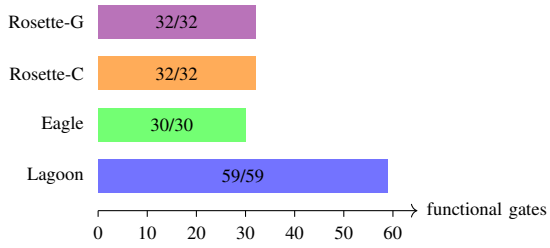

Fig. 8. Nested functional-gate coverage in the fresh Stage150 artifact. Counts are assertion coverage, not accuracy samples.

0.0091. In matched experiments reported in the first study, the Civilization path, MLP adapter, LoRA, and prefix tuning all saturated the controlled benchmark. Accordingly, this paper uses Stage42 only to establish the inherited path and frozen-base contract, not predictive superiority.

D. Negative Results and Claim Boundary

Several findings constrain interpretation. Earlier Sun real-task migration did not establish transfer on public tasks; projected path readout could pass while raw hidden-state readout remained near chance; and a nonzero delta was not sufficient for answer-direction correctness. In the memory versions, retrieval uses lexical overlap rather than semantic embeddings, consolidation policies are hand-coded, and release gates use controlled fixtures. The successful results therefore concern invariants, traceability, persistence, and controlled integration, not general intelligence or lifelong adaptation.

XIX. DISCUSSION

A. A Model–System Architecture

The central design choice is separation of evidence from policy. An episode is evidence that an event occurred; a semantic cell is an approved abstraction; a procedural cell is an approved strategy; and a conflict decision is control state. Keeping these objects distinct makes provenance queries and

policy changes possible without rewriting historical memory. The resulting artifact is neither an external database attached to an otherwise unchanged chatbot nor a newly pretrained foundation model. It is a model–system architecture: the runtime, Adapter path, memory types, transition policies, persistence, and audit interfaces jointly define behavior.

The current release keeps Qwen3 frozen because that choice isolates memory semantics from base-model drift. It should not be interpreted as a permanent architectural commitment. A future training regime may jointly optimize the retrieval router, context projection, consolidation policy, and neural core, provided the typed evidence and intervention contracts remain testable.

B. Auditability as an Operational Property

Auditability here is not a visualization added after prediction. It is the ability to reconstruct which cell was selected, which policy admitted or excluded it, which source records support a derived object, which control changed the Adapter path, and which artifact generation was loaded. This is why the architecture preserves rejected and conflicting evidence instead of only recording the final response. The approach increases storage and policy complexity, but it permits later rules to be evaluated against the same historical evidence.

The architecture also separates memory correctness from model benefit. A perfectly traced store can retrieve irrelevant information, while a useful model intervention can be difficult to audit. The staged protocol therefore tests store invariants, context assembly, and hidden-path intervention independently. Future accuracy experiments can replace the lexical scorer or candidate policies while retaining the same cells, links, traces, and gates.

C. Approval and Conservative Mutation

Approval identifiers are software preconditions, not evidence that an organizational governance process is complete. Their current value is to make the mutation boundary explicit and testable. Candidate generation can be repeated, inspected, or replaced without changing semantic or procedural memory; mutation occurs only at the approved transition. In deployment, the identifier would need an authenticated principal, policy version, reason, and durable authorization record.

The honeycomb graph should be understood as a typed projection rather than a new geometric learning primitive. Its value in the current version is that a query can traverse support, conflict, and decision relations under explicit budgets. Learning edge weights is future work and would require held-out tasks, baselines, and calibration checks.

D. Why the Staged Method Matters

The stages are intentionally narrower than paper-level contributions. This granularity made it possible to reject a failed task-recovery route, distinguish candidate evidence from approved memory, and test snapshots separately from live topology. The cost is a long implementation history that can look fragmented when summarized only by release names. The complete stage map makes the dependency structure explicit: each stage either adds one object, closes one failure boundary, exposes one integration surface, or freezes a release gate.

XX. Limitations and Threats to Validity

External validity. Most v0.00.02–v0.00.06 evidence comes from small, deterministic fixtures designed to expose contract violations. It does not measure long-horizon user tasks, open-domain retrieval, latency under load, or cross-model generality.

Historical versus fresh evidence. The complete Sun stage map is an implementation history, while the fresh paper reproduction begins with Orion and includes one real Stage75 integration diagnostic. The full Stage44–72 service and deployment chain was not rerun as one new paper benchmark.

Retrieval quality. Equation (3) uses token overlap and hand-set coefficients. It is reproducible and inspectable but weak under paraphrase, multilingual variation, and large stores. No vector database or learned retriever was evaluated.

Consolidation validity. Episode signatures and clustering are symbolic policies over supplied fields. A passing gate does not show that the resulting schema is epistemically correct. Human or trusted-policy approval is still required, and adversarial source provenance has not been evaluated.

Model evidence. The fresh CUDA diagnostic uses one Qwen3-0.6B request and demonstrates path activation, zeroing, and frozen weights. It is not an accuracy study. The inherited three-seed Sun result is confined to a local five-candidate benchmark with a ceiling effect among trained baselines.

Scale and cost. The implementation does not establish retrieval or graph behavior at large cell counts. Storage growth, trace compaction, concurrent memory mutation, tail latency, and recovery time need dedicated load tests. The reported service tests are contract fixtures rather than an operational service-level objective.

Coverage interpretation. The 153 functional assertions measure explicit release conditions, not independent samples and not formal proof. Several gates share fixture builders and upstream objects, so their count must not be interpreted as statistical evidence.

Operational security. Snapshot integrity hashes detect accidental or unsophisticated mutation; they are not digital signatures. The research artifact omits production authentication, encryption, multi-tenant isolation, and key management.

Disclosure boundary. The public package omits private payloads, configuration values, model weights, and deployment details. This protects the operational boundary but prevents a third party from reproducing the complete private environment from the paper package alone.

Biological interpretation. Terms such as hippocampal-style binding and complementary systems describe functional inspiration only. The software does not model hippocampal physiology, neural replay dynamics, or human memory capacity.

XXI. Conclusion

This report presented the complete public v0.00.01–v0.00.06 evolution of an auditable memory architecture, from

a frozen-model runtime substrate through typed multi-system memory, episode binding, approved semantic consolidation, trace-backed procedures, atomic context groups, and multi-scale graph views. The Stage44–150 map shows how operational capabilities and negative controls became dependencies for later memory versions rather than unrelated feature additions.

The main technical result is a set of executable invariants: rapid binding preserves sources; replay precedes candidate formation; approvals gate semantic and procedural writes; conflicts preserve both facts; decisions remain separate from factual context; context budgets keep preserve-both groups atomic; and snapshots fail closed under corruption or topology mismatch. Fresh WSL and CUDA reproduction confirmed the current test suites, artifact chain, frozen-Qwen integration, and no-memory intervention within the reported fixtures. The evidence supports a versioned systems foundation for future memory-model training and evaluation. Establishing improved long-horizon task performance, learned consolidation, operational scale, and robust multi-model transfer remains future work.

APPENDIX A
REPRODUCIBILITY AND EVIDENCE LEDGER

The accompanying package contains the exact fresh test logs, local and WSL source manifests, Stage75 diagnostic outputs, the complete fresh Stage150 artifact tree, and a machine-readable evidence audit. The public manuscript uses repository-relative identifiers and excludes host addresses, credentials, private keys, and model filesystem locations.

The principal reproduction sequence is: (1) compare SHA-256 manifests; (2) run the three staged pytest groups; (3) execute Stage75 on CUDA into a new directory; (4) execute Stage150 into a new directory; (5) run `scripts/verify_evidence.py`; and (6) compile and visually inspect the PDF. The evidence script recomputes test counts from logs, validates all seven release gates, checks graph scale/adjacency coverage, verifies the Stage75 intervention, and writes `evidence/reproduction-summary.json`.

APPENDIX B
MEMORY-SYSTEM SEMANTICS

Working memory is the only system with mandatory TTL semantics. Episodic cells are ordered event anchors and remain provenance sources. Semantic and procedural cells are derived only through explicit transitions. This policy is stricter than the data structure itself: the common cell schema permits shared fields, while stage gates constrain which transitions are valid.

The public source package contains the ledger, source manifests, runner logs, and verification script. It intentionally does not include model weights, private execution paths, credentials, or host configuration. Reproduction of the artifact contracts is therefore separable from redistribution of the frozen model.

TABLE XIX
MEMORY-SYSTEM LIFECYCLE SEMANTICS

System	Lifetime and read boundary	Derived role
Working	TTL; active by default	episode source candidate
Episodic	ordered/bound/replayed; time, scene, source, and cue filters	semantic or skill evidence
Semantic	persistent and conflict-aware; provenance/decision filters	context fact
Procedural	persistent and outcome-backed; task/provenance filters	control or action policy

REFERENCES

[1] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly, "Why there are complementary learning systems in the hippocampus and neocortex," *Psychological Review*, vol. 102, no. 3, pp. 419–457, 1995.

[2] D. Kumaran, D. Hassabis, and J. L. McClelland, "What learning systems do intelligent agents need? complementary learning systems theory updated," *Trends in Cognitive Sciences*, vol. 20, no. 7, pp. 512–534, 2016.

[3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Kuttler, M. Lewis, W.-t. Yih, T. Rocktaschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems*, 2020.

[4] J. Weston, S. Chopra, and A. Bordes, "Memory networks," *arXiv preprint arXiv:1410.3916*, 2014.

[5] A. Pritzel, B. Uria, S. Srinivasan, A. P. Badia, O. Vinyals, D. Hassabis, D. Wierstra, and C. Blundell, "Neural episodic control," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, 2017, pp. 2827–2836.

[6] P. W. Battaglia, J. B. Hamrick, V. Bapst *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[7] Y. Wang, "Civilization architecture: Structured memory, state, and rule pathways for auditable hidden-state computation," technical preprint, 2026.

[8] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. van den Driessche, J.-B. Lespiau, B. Damoc, A. Clark *et al.*, "Improving language models by retrieving from trillions of tokens," in *International Conference on Machine Learning*, 2022.

[9] S. Sukhbaatar, J. Weston, R. Fergus *et al.*, "End-to-end memory networks," in *Advances in Neural Information Processing Systems*, 2015.

[10] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[11] B. J. Gutiérrez, Y. Shu, Y. Gu, M. Yasunaga, and Y. Su, "HippoRAG: Neurobiologically inspired long-term memory for large language models," in *Advances in Neural Information Processing Systems*, vol. 37, 2024.

[12] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023, pp. 1–22.

[13] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg *et al.*, "Graph networks as learnable physics engines for inference and control," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, 2018, pp. 4470–4479.

[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[15] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. de Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *International Conference on Machine Learning*, 2019.

[16] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022.

[17] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," in *Annual Meeting of the Association for Computational Linguistics*, 2021.