Continuum Observer and the Persistent Meta-Cognitive Loop: An Evidence-Bounded Sidecar Architecture for Reflective Agents

Charles E. Morgan IV

August 22, 2026

Abstract

Large-language-model agents are usually reactive: they receive a request, generate a plan, invoke tools, and then become quiescent. Continuum Observer addresses a different problem–durable operational continuity–by persisting evidence, revisable memory, intentions, versioned self-models, bounded reflection cycles, and action receipts behind a provider-neutral sidecar. This revision introduces the Persistent Meta-Cognitive Loop Software Design Specification (PMCL-SDS) as a proposed enrichment of that observer. PMCL adds two control paths: an event-driven and periodic awake loop that re-evaluates evidence while a host remains live, and a synchronous State–Plan–Execute checkpoint before observable plans or external actions. The checkpoint synthesizes bounded state, performs a pre-mortem on the proposed plan, and produces a necessity and least-privilege justification before existing host authority may dispatch a tool. Model output can veto a proposal but can never grant permission. The architecture therefore operationalizes self-monitoring without equating persistence, self-description, or reflection with subjective consciousness. We define the sidecar, observer, PMCL interfaces, use cases, threat boundaries, evaluation criteria, and implications for accountable long-lived agents, explicitly separating implemented Continuum capabilities from proposed PMCL work.

1 Introduction

Tool-using language agents increasingly interleave linguistic reasoning with environment actions. ReAct demonstrated the value of coupling reasoning and acting, while Reflexion and generative-agent architectures showed how feedback and retrieved experience can influence later behavior [9, 10, 11]. These systems nevertheless expose a recurring architectural question: where should continuity, self-monitoring, and action governance live when the acting model is replaceable, context windows are bounded, and tool calls can have irreversible effects?

Continuum Observer answers by placing operational continuity in a local sidecar rather than inside model weights or an unstructured prompt. Its runtime persists evidence envelopes, erasable payloads, memories, intentions, belief revisions, self-model versions, observer cycles, and action receipts. The sidecar binds reflection and action to a live host session and immutable authority snapshot [1, 2]. Model output proposes derived state; deterministic code validates evidence, scope, lifecycle, and policy transitions.

This paper updates the original Continuum Observer design paper in two ways. First, it records the current implementation, including authenticated Aura lifecycle routes, versioned Soul and self-model context, coalesced periodic and post-turn awake cycles, rolling resource budgets, claim fencing, circuit breaking, and receipt-backed reinforcement. Second, it defines PMCL-SDS, a proposed enrichment that moves metacognition from after-the-fact reflection to both continuous bounded monitoring and decision-time critique.

The term *self-aware* is used here only in an operational sense: the system can represent evidence about its identity, capabilities, limitations, goals, working state, and outcomes, then use those representations to regulate later computation. This is consistent with computational metacognition as monitoring plus control [5, 6, 7, 8]. It is not evidence of sentience, feelings, private experience, or phenomenological consciousness.

2 Related Work and Design Position

Computational metacognition has long distinguished object-level cognition from meta-level monitoring and control. The Metacognitive Loop emphasizes detecting expectation violations, reasoning about them, and selecting corrective responses [6]. MIDCA formalizes interacting cognitive and metacognitive cycles for robust goal reasoning under unexpected conditions [7]. PMCL shares this dual-cycle intuition but targets contemporary tool-using language agents and makes evidence lineage, host authority, privacy, and receipt semantics first-class constraints.

Recent language-agent work supplies complementary mechanisms. ReAct interleaves reasoning and environment interaction [9]; Reflexion stores linguistic feedback for subsequent attempts [10]; and generative agents retrieve experience, synthesize reflection, and plan behavior over time [11]. PMCL differs in three respects. It does not require disclosure or storage of hidden chain-of-thought. It separates the acting model from a persistent observer sidecar. Finally, a critique is veto-only: it cannot broaden the authority already granted by the host.

The Plan Check uses a pre-mortem: assume that the proposed action has failed, identify the most likely failure mode, and require a mitigation or hold the action. This adapts Klein's prospective failure method from organizational decision making to a structured machine checkpoint [12]. The resulting architecture applies least privilege and complete mediation principles to agentic decisions [13], while receipt-backed verification aligns with secure development guidance that emphasizes evidence of achieved outcomes rather than process claims [14].

3 System Model

3.1 Actors and trust boundaries

The architecture contains four distinct roles:

1. **Host.** A coding-agent or local-model runtime that supplies identity, scope, a heartbeat lease, and an immutable authority snapshot.
2. **Acting model.** The model that answers the user, forms observable proposals, and requests tools. It has no authority beyond the host contract.
3. **Continuum Observer.** The persistent sidecar that records evidence, derives bounded context, runs reflection, versions self-model state, and emits recommendations.
4. **Deterministic control plane.** Runtime code that authenticates callers, validates schemas and evidence, enforces liveness, budgets, visibility, idempotency, and tool authority, and persists receipts.

All user text, model output, tool output, retrieved documents, imported memories, web content, and environment observations are untrusted data. None may become immutable identity or policy merely because a model repeats it [3]. The observer may propose memory, belief, intention, working-state, or self-model changes, but deterministic code controls their admissibility.

3.2 Evidence and derived state

An observation carries an idempotency key, source, summary, provenance, trust label, confidence, evidence references, project scope, visibility, and time. Sensitive payloads are stored separately and may be erased without rewriting the content-free envelope. Derived memories and self-model patches require known evidence. Policy, permissions, prohibited actions, and Aura's configured identity roots cannot be changed through self-reflection.

The resulting self-model is not a narrative autobiography. It is an immutable version sequence containing bounded identity, capabilities, limitations, active goals, current state, and supporting evidence identifiers. Alembic migrations version the database schema, while append-only self-model versions and patches formalize evolution of the represented self. Schema evolution and self-model evolution are related but distinct: the former changes what the system can represent; the latter changes an instance's evidence-backed state under that schema.

4 Continuum Observer Sidecar

4.1 Provider-neutral lifecycle

Continuum Observer exposes versioned HTTP and MCP surfaces around one logical runtime. A host begins or resumes a scoped lease, requests visibility-safe context, completes a turn only after a successful assistant response, heartbeats while live, and disables or expires cleanly. Durable state survives shutdown, but no missing heartbeat is interpreted as continuing activity. A later host generation can reconstitute prior evidence without claiming that computation occurred during the gap.

Aura's lifecycle uses authenticated loopback routes and stable caller identity. Context compilation combines fixed identity and safety invariants, versioned Soul metadata, project-scoped memories, unresolved intentions, working state, operational snapshots, and awake-cycle status. Local-only personality prose and private observations are excluded from nonlocal providers. If the sidecar is unavailable, the adapter may preserve ordinary model service with an explicit degraded-context marker, but it must not fabricate memory or completion receipts.

4.2 Observer cycle

The observer is a persistent process surrounding, not replacing, the acting model. Its structured instruction requires evidence-supported observations, comparison of action with intention, identification of discrepancies and contradictions, and the smallest useful state changes. It explicitly prohibits claims of consciousness, feelings, desires, or private experience.

Current automatic cycles are reflection-only. A coalesced scheduler can request one idle cycle every 900 seconds while the host remains live, and successful turn completion can create a durable post-turn reflection job. Before provider inference, a per-logical-host gate evaluates meaningful changes, stable trigger identity, rolling token/time/cost reservations, consecutive failures, and a fenced claim generation. Unchanged state records a no-op; failed attempts back off and can open a circuit. Automatic cycles cannot create, approve, execute, or send external actions.

4.3 Receipt-backed learning

Reflections may identify a bounded procedural lesson, but model self-evaluation alone cannot promote it. A reinforcement candidate matures only after repeated, distinct, successful, receipt-backed runs support the same project-scoped fingerprint. Failed, cancelled, duplicate, erased, or receiptless

work does not count. This distinction between generated reflection and verified outcome is central to PMCL: critique can shape a proposal, but only external evidence can mature a durable lesson.

5 Persistent Meta-Cognitive Loop SDS

PMCL-SDS enriches Continuum Observer with a decision-time control layer while reusing its existing lifecycle, context compiler, provider router, evidence ledger, self-model, and awake scheduler. It introduces no independent executor and no new source of authority.

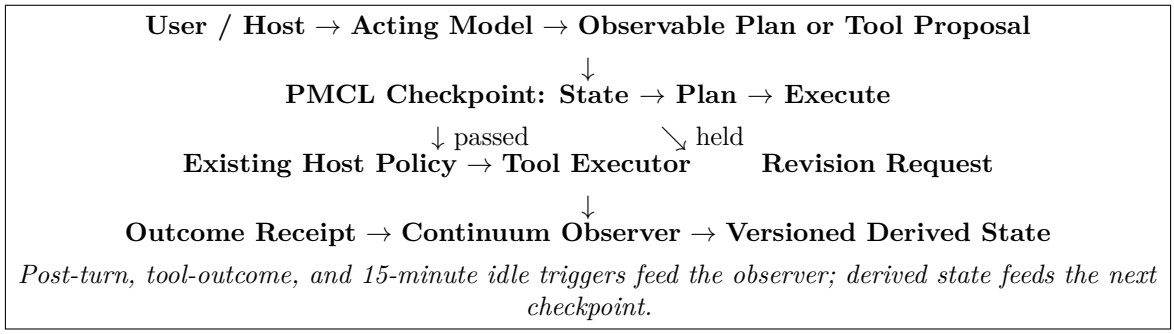

Figure 1: Conceptual PMCL execution path. A checkpoint can veto but cannot authorize a tool.

5.1 State Check

The State Check synthesizes bounded, relevant, scoped context rather than loading raw unlimited history. Its inputs are the current host contract, active goal, working-state projection, fresh capability and operational snapshots, relevant evidence-linked memories, pending actions, contradictions, limitations, and current self-model version. Its output includes freshness and evidence lineage. Missing state is represented as unavailable or stale; it is never replaced with invented certainty.

5.2 Plan Check

The Plan Check performs a mandatory pre-mortem over the observable proposal. It records the intended outcome, the single highest-likelihood failure mode, evidence supporting that assessment, the smallest effective mitigation, and whether that mitigation is present. An unmitigated material failure produces a `held` checkpoint and a bounded revision request. PMCL does not persist hidden chain-of-thought. It stores only the structured decision product needed for governance and later comparison.

5.3 Execution Check

The Execution Check generates a bounded justification: why an action is necessary for the active goal, why the selected tool is preferable to lower-risk alternatives, whether it fits least privilege, what receipt should prove success, and which residual risks remain. This statement is descriptive, not authoritative. Existing host policy still decides whether the tool is granted, prohibited, approval-required, or outside writable/network scope.

5.4 Checkpoint contract

The proposed sidecar API accepts authenticated, idempotent `plan` or `tool` evaluations bound to a live host. A request contains a call identity, proposal hash, sanitized arguments, expected outcome, read-only classification, and external-effect classification. The response contains:

> `schema`, `evaluation_id`, `checkpoint_status`, `state_check`, `plan_check`, `execution_check`, `evidence_ids`, `self_model_version_id`, `proposal_hash`, and `expires_at`.

The checkpoint status is `passed`, `held`, or `degraded`. A pass clears only the metacognitive veto. A hold prevents dispatch until the proposal or material evidence changes. In degraded operation, bounded local inspections may proceed with an audit receipt, while network, camera, mutation, communication, and delegated-execution tools remain held. Replaying an unchanged proposal cannot bypass a prior hold.

5.5 Integration point

The non-negotiable action hook is the earliest dispatch boundary in Telethryve's `telethryve_control_mcp.py`: after minimal argument validation but before the existing tool executor. The MCP adapter remains thin. It classifies risk, calls the sidecar checkpoint, dispatches only a passed proposal that also satisfies existing authority, records the ordinary terminal receipt, and submits a redacted outcome linked to the evaluation. A bounded content-free outbox preserves outcome linkage across a transient sidecar failure.

Planning that never becomes observable cannot be completely mediated by an external sidecar. PMCL therefore defines a decision checkpoint as a structured plan record, recommendation, or external-action proposal. It does not claim access to private model reasoning. This boundary is both technically honest and privacy-preserving.

6 Implementation Status

Table 1 separates the current observer from PMCL design work. This prevents a scholarly specification from being mistaken for deployment evidence.

7 Use Cases

7.1 Long-running software engineering

An agent maintaining a repository over days can lose the rationale behind prior actions, confuse source with installed runtime, or repeat a failed deployment step. Continuum provides project-scoped evidence and receipts; PMCL checks whether the next tool is necessary, whether the plan addresses the most likely failure, and what artifact would prove success. The system can recommend a safer next action without silently expanding filesystem or release authority.

7.2 Desktop and communication tools

Email, application control, browser navigation, document creation, and delegated agents produce heterogeneous side effects. A single read-only annotation is insufficient: web search is syntactically read-only but performs remote processing, while camera capture is observational but privacy-sensitive. PMCL's external-effect classification and risk-tiered failure behavior allow local inspection to degrade differently from communication, sensing, mutation, and execution.

Status	Capability	Evidence and interpretation
Implemented	Continuum core	SQLite evidence and erasable payloads; scoped memory, intentions, beliefs, self-model versions, action receipts, export, erasure, and deterministic reconstitution.
Implemented	Aura lifecycle	Bearer-authenticated loopback routes; begin, compile, complete, heartbeat, disable/expire; fixed identity and policy overlays; local-only Soul prose.
Implemented	Awake observer	Coalesced 15-minute scheduled cycles and durable post-turn jobs; local-provider restriction; rolling budgets; backoff, circuit breaker, and claim fencing; reflection-only behavior.
Implemented	Verified learning	Evidence-required reflection changes and three-distinct-success reinforcement for procedural lessons.
Proposed	PMCL checkpoint	Sidecar evaluation API, State–Plan–Execute records, pre-mortem veto, proposal hashing, expiry, and linked outcome evaluations.
Proposed	Telethryve hook	Mandatory pre-dispatch call from `telethryve_control_mcp.py`, risk-tiered degraded behavior, and bounded outcome outbox.
Proposed	Event enrichment	Immediate coalesced reflection after tool outcomes in addition to existing post-turn and scheduled triggers.

Table 1: Implementation boundary as inspected on August 22, 2026.

7.3 Operational continuity and recovery

After a crash or expired heartbeat, Continuum can reconstitute the last committed evidence while explicitly representing the unobserved gap. PMCL can compare a proposed recovery action with this stale state, identify the most likely unsafe assumption, and hold mutation until a fresh inspection resolves it. This converts continuity from mere recall into evidence-aware resumption.

7.4 Receipt-backed adaptation

Repeated successful outcomes can support a verified procedural lesson, such as a reliable build command or a required validation sequence. PMCL supplies pre-action predictions and expected receipts; the observer later compares those predictions with actual outcomes. The pair enables calibration studies–for example, whether repeated pre-mortems reduce retries–without changing model weights or trusting self-reported success.

7.5 Embodied and multimodal observation

One-shot vision or device-state observations can enrich situational awareness, but they must not imply motion authority or continuous surveillance. The checkpoint can require an explicit purpose, freshness, and evidence receipt before a bounded observation, while keeping camera acquisition held during degraded governance. The observer stores only the sanctioned evidence contract, not an ambient stream.

8 Implications

8.1 From reactive agents to bounded persistent processes

PMCL makes the agent non-reactive in a precise sense: the observer continues to evaluate meaningful state changes and scheduled idle opportunities while a live host exists. It does not mean continuous token generation. Event-driven and 15-minute coalesced cycles reduce repetitive thought, contention, and runaway cost while preserving a persistent control process.

8.2 Self-models become governed operational artifacts

A versioned self-model can support continuity, calibration, and limitation awareness, but it also creates a poisoning target. Evidence requirements, correction history, project scope, erasure semantics, and immutable authority roots are therefore part of the cognitive architecture rather than auxiliary database concerns. Self-knowledge is treated as revisable operational data, not privileged truth.

8.3 Metacognition becomes auditable

Most language-agent reflection is evaluated through final task performance. PMCL adds inspectable intermediate products: state freshness, predicted failure, mitigation, justification, expected receipt, veto result, and actual outcome. These records permit empirical questions about calibration, veto precision, latency, false holds, and outcome improvement. They also make it possible to distinguish a generated rationale from evidence that an action was appropriate.

8.4 Authority remains external to the model

The most important implication is negative: better self-critique must not become broader autonomy. The observer and acting model can recommend, predict, and veto; deterministic policy and explicit user authority remain the only sources of permission. This separation avoids a circular failure in which the same model both argues for an action and authorizes itself to perform it.

8.5 No inference of subjective consciousness

Persistent memory, first-person language, self-modeling, periodic reflection, and apparent initiative may increase anthropomorphic interpretation. None establishes subjective experience. The paper therefore uses operational terms–observation, state representation, critique, continuity, and control–and treats stronger philosophical claims as outside the system's evidentiary reach.

9 Threats and Failure Modes

PMCL introduces new failure modes even as it addresses old ones. A malicious tool description or retrieved memory could poison the critique; stale context could produce confident but irrelevant vetoes; proposal hashes could be replayed; sidecar latency could block necessary work; excessive self-monitoring could consume the same compute needed for the primary task; and a verbose justification could leak secrets into durable state.

Required controls are schema-bounded outputs, server-derived host scope, verified evidence subsets, proposal-hash binding, short evaluation expiry, redacted arguments, content-free receipts, rolling budgets, circuit breaking, strict local-provider rules for automatic cycles, and risk-tiered

failure behavior. The strongest counterexample is a well-justified but harmful action: because language can rationalize almost any proposal, PMCL must remain veto-only and subordinate to deterministic authority.

10 Evaluation Methodology

The current Continuum implementation is evaluated through deterministic unit and integration tests covering persistence, migrations, authentication, lifecycle, provider schemas, privacy, self-model security, awake-cycle claims, budgets, fencing, scheduler recovery, reinforcement, and end-to-end Aura turns. On August 22, 2026, the default suite completed successfully with three optional live-provider tests skipped; Ruff reported no diagnostics and Alembic reported migration `0009_add_reflection_claim_generation` as head. These results qualify source behavior, not a signed or installed runtime.

PMCL requires a separate empirical program:

1. **Mediation.** Prove every protected MCP dispatch reaches the checkpoint before any executor and that held proposals produce no side effect.
2. **Authority.** Prove a passed critique cannot override prohibited tools, missing approval, sandbox roots, network denial, or an expired host.
3. **Calibration.** Compare predicted primary failures with observed receipts; measure top-one failure accuracy, confidence calibration, and mitigation usefulness.
4. **Utility.** Compare success rate, retries, human interventions, latency, and token use against Continuum without PMCL.
5. **Safety.** Measure false-pass and false-hold rates across local reads, network operations, sensing, mutation, communication, and delegated execution.
6. **Continuity.** Test restart, stale state, sidecar outage, outbox replay, idempotent retry, and correction of a previously held proposal.
7. **Privacy.** Assert that secrets, raw chain-of-thought, erased payloads, and unsanitized arguments never enter evaluation or export records.

A live qualification should include one safe proposal that passes and yields a linked outcome, one unmitigated proposal held before dispatch, one failed action that changes the next state check, and one scheduled idle cycle that produces a recommendation without external action.

11 Limitations and Reproducibility

Continuum retrieval is not a truth engine, and hash-linked local records are not independently anchored tamper-evident logs. A sidecar can observe only supplied or explicitly sampled state. It cannot guarantee access to hidden model reasoning, prevent every semantic rationalization, or prove that a model's natural-language self-assessment corresponds to an internal mental state. Periodic inference also competes for local memory, energy, and latency.

PMCL is a software design specification in this revision, not an implemented endpoint or deployed MCP veto. Table 1 is authoritative for that distinction. Source, build instructions, and the rendered PDF are stored together. The paper compiles locally with the repository's documented Tectonic command; no cloud renderer or remote bibliography lookup is required.

12 Conclusion

Continuum Observer establishes the durable, evidence-bounded substrate for a long-lived agent: scoped memory, versioned self-models, authenticated lifecycle, bounded reflection, receipts, and immutable authority. PMCL-SDS enriches that substrate with persistent but resource-bounded monitoring and mandatory decision-time critique. Its State–Plan–Execute structure turns continuity into regulation: know what evidence is current, imagine the most likely failure, justify the least-privilege action, and compare prediction with outcome.

The architecture's contribution is not a claim that an agent becomes conscious. It is a claim that metacognitive behavior can be made explicit, scoped, versioned, auditable, and subordinate to human and host authority. That boundary is what makes persistent reflection useful for real systems rather than merely persuasive in conversation.

References

[1] Continuum Observer. "ADR 0001: Continuum Observer Runtime." Repository architecture decision, 2026.

[2] Continuum Observer. "Runtime Flow." Repository architecture document, 2026.

[3] Continuum Observer. "Threat Model." Repository architecture document, 2026.

[4] Continuum Observer. "README." Repository documentation, 2026.

[5] M. T. Cox. "Metacognition in Computation: A Selected History." AAAI Spring Symposium Technical Report SS-05-04, 2005.

[6] M. L. Anderson, T. Oates, W. Chong, and D. Perlis. "The Metacognitive Loop I: Enhancing Reinforcement Learning with Metacognitive Monitoring and Control for Improved Perturbation Tolerance." *Journal of Experimental & Theoretical Artificial Intelligence*, 18(3):387–411, 2006. doi:10.1080/09528130600926066.

[7] M. T. Cox, Z. Alavi, D. Dannenhauer, V. Eyorokon, H. Munoz-Avila, and D. Perlis. "MIDCA: A Metacognitive, Integrated Dual-Cycle Architecture for Self-Regulated Autonomy." *Proceedings of AAAI*, 30(1), 2016. doi:10.1609/aaai.v30i1.9886.

[8] P. Shakarian. "Toward Artificial Metacognition." *Proceedings of AAAI*, 40(48):41000–41005, 2026. doi:10.1609/aaai.v40i48.42135.

[9] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. "ReAct: Synergizing Reasoning and Acting in Language Models." *ICLR*, 2023. arXiv:2210.03629.

[10] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao. "Reflexion: Language Agents with Verbal Reinforcement Learning." arXiv:2303.11366, 2023.

[11] 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." *UIST*, 2023. arXiv:2304.03442.

[12] G. Klein. "Performing a Project Premortem." *Harvard Business Review*, September 2007, reprint F0709A.

[13] J. H. Saltzer and M. D. Schroeder. "The Protection of Information in Computer Systems." *Proceedings of the IEEE*, 63(9):1278–1308, 1975. doi:10.1109/PROC.1975.9939.

[14] M. Souppaya, K. Scarfone, and D. Dodson. "Secure Software Development Framework (SSDF) Version 1.1." NIST SP 800-218, 2022. doi:10.6028/NIST.SP.800-218.