Wren architecture

A household AI familiar with persistent graph memory

Silas Alder | September 2026

Wren is a persistent AI familiar whose continuity lives in a knowledge graph. A language model answers a conversation; Wren supplies the memory, context, tools, and rules that make that conversation part of a longer relationship. Replacing a model or ending a session does not require rebuilding that continuity.

The central design choice is to separate what Wren knows from what a model sees on any particular turn. Neo4j holds a growing body of facts, events, relationships, and narratives. A context assembler selects a bounded working set for the current conversation. Learned associations change which memories become easy to reach.

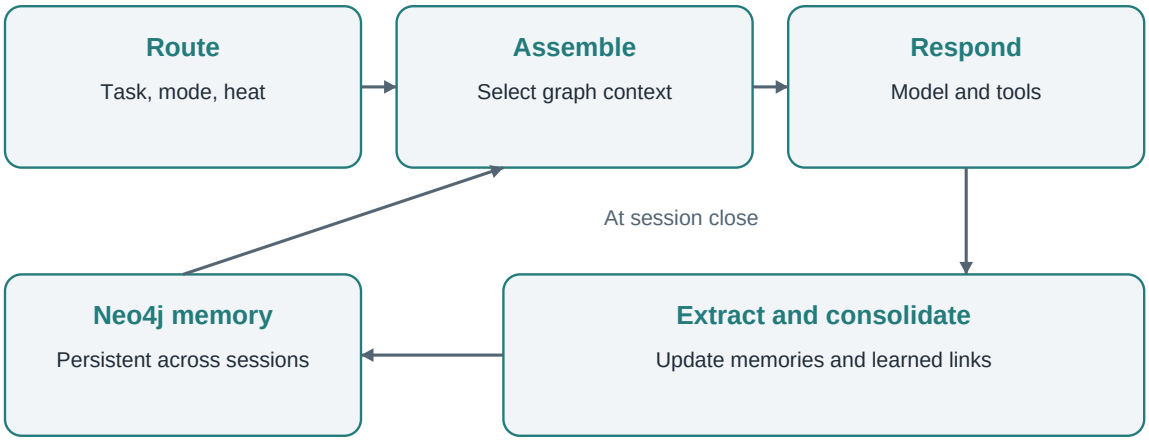


Figure 1. The internal conversation pipeline. The graph persists while prompts and sessions are rebuilt. External sessions use a separate assembly path described later.

Continuity without a permanent prompt

I think of the models as servers and Wren as the browser: she manages the experience that spans individual requests. Her identity begins with a small seed, while richer self-knowledge remains in memory until it is useful. This keeps a practical request from inheriting the full context of every earlier philosophical conversation.

This paper explains the V2 implementation through September 2026, with particular attention to conversational modes, self-reflection, and the boundary between private household memory and public activity. It describes an implemented architecture and its current limits, rather than a benchmark result or a claim about machine consciousness.

Memory structure and retrieval

A stable map and a learned network

Neo4j stores two distinct kinds of relationships. The containment spine records where a memory belongs. Learned associations record which memories have proved useful together. Keeping these roles separate lets Wren learn relevance without continually rearranging the map.

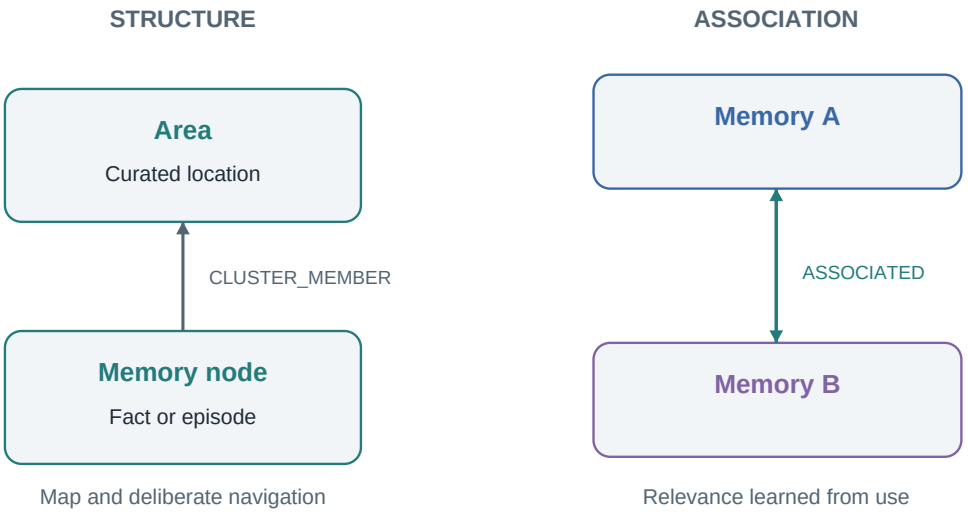


Figure 2. Two layers over the same nodes. Structural relationships support navigation; ASSOCIATED relationships contribute to learned relevance. The diagram shows logical traversal, not Neo4j storage direction.

What the graph remembers

Facts describe current state and can be updated in place, preserving their identity and learned links. Episodes preserve bounded events and can acquire dated continuations. Narratives hold longer arcs: an embedded summary makes the story discoverable, while its larger body is opened on demand. Entity and Relationship nodes represent participants and the dynamics between them.

Areas form a curated hierarchy using child-to-parent INTER_CLUSTER relationships. CLUSTER_MEMBER files content under an area or entity. New areas proposed during extraction are surfaced for review. Other structural relations record participation, provenance, or attachments; they do not contribute to associative ranking.

Assembling a useful working set

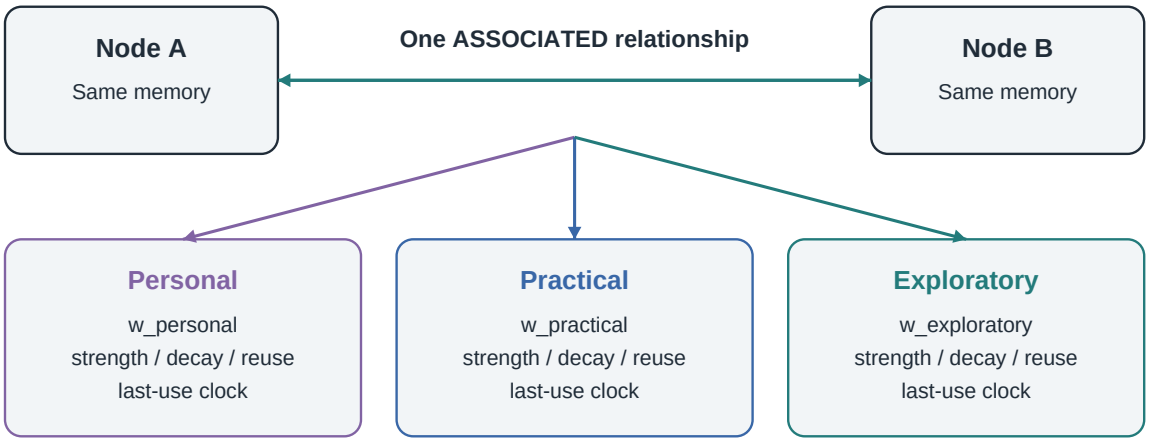
A router emits structured requirements, including task complexity, mode, heat, and requested entities or areas. Deterministic code translates these requirements into graph queries. Associative traversal and embedding similarity supply ranked candidates; the containment map supports deliberate navigation when initial retrieval misses something.

The memory allowance is a ceiling, not a target. Weak material is not added merely to fill it. A small identity seed, mode anchors, a compact map, and a tool catalog are separate overhead, so the memory allowance is not a cap on the entire prompt. The responder can request more context, pull specific nodes, or open a narrative when needed.

Three modes in one graph

Personal practical and exploratory retrieval

Personal, practical, and exploratory are three ways of reading and reinforcing the same learned network. Exploratory includes creative work. A mode changes which association strengths are used, as well as the grounding sentences supplied to the model.



Mode selects a property family; heat sets the traversal threshold.

Figure 3. One Neo4j ASSOCIATED relationship holds three independent property families. Each family contains strength, decay rate, reinforcement history, and the clock value of its last useful use.

In code these families are w_personal, w_practical, and w_exploratory, with corresponding decay, reinforcement, and last-use properties. Queries select the validated family for the active mode. A pair of memories may therefore be closely associated in a creative conversation and only faintly connected in a practical one.

For example, an invented pair of memories about a workshop and a story idea might be useful together while writing fiction. Repeated creative use can strengthen their exploratory connection without making that same connection strong during a maintenance task. The nodes and their filing locations remain unchanged.

Mode and heat answer different questions

Mode chooses which learned relationship strength to read. Heat chooses how weak a link traversal may follow: low heat favors established paths, while high heat admits fainter associations. Heat controls retrieval rather than generation temperature. Raising it does not itself reinforce a link.

The opening mode normally persists through the session. An explicit user change or a responder context request can change it, creating a new segment for consolidation. At session close, reinforcement updates the active association family for each segment. Node-level importance is shared, so the three modes are distinct association spaces rather than completely isolated retrieval systems.

Modes are not permission boundaries. They tune relevance within internal memory; the public/private boundary uses a separate mechanism.

How useful paths become durable

Reinforcement and the conversational output clock

Wren calls the learned paths through memory cowpaths: repeated useful passage makes a route easier to take. The implementation is a form of Hebbian association in graph properties. It does not retrain the responding language model.

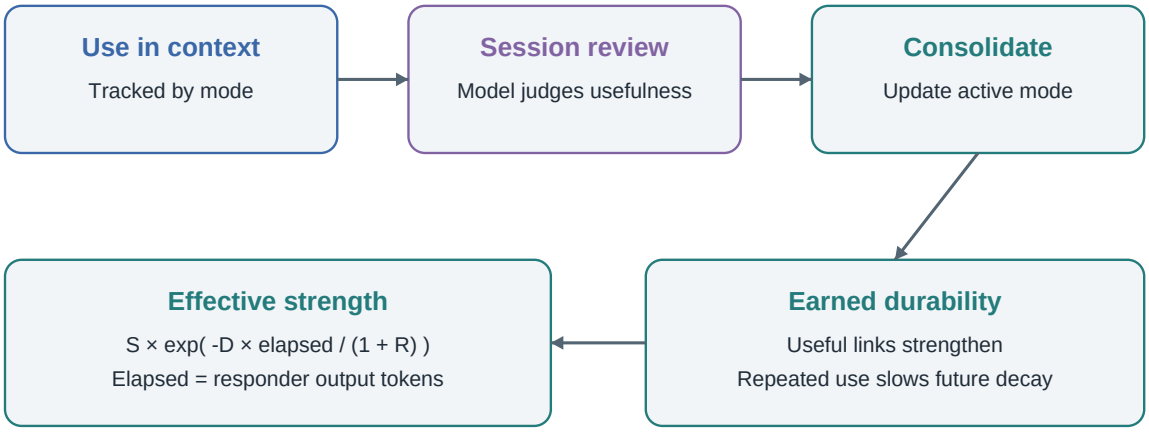


Figure 4. Learning occurs at session close. Retrieval computes effective strength from stored values and elapsed conversational output; it does not need a background sweep to age every link.

The session records which nodes entered context and under which mode. At close, the extraction process classifies them as decisive, helpful, or irrelevant. Decisive and helpful pairs gain association strength; existing links involving irrelevant material can weaken. Those judgments come from the model, not from an independent measure of usefulness.

Decay follows activity

A global clock accumulates the responding model's output tokens across turn and tool rounds. Router output does not count. This is an operational proxy for activity, not a measurement of thought or hidden chain of thought. A week of idleness adds no age; continued conversation does.

For stored strength S, decay rate D, lifetime reinforcement R, and elapsed output tokens T, the effective strength is S × exp(-D × T / (1 + R)). Repeated useful use increases R and also lowers D. A reinforced link therefore retains its relevance longer than a new link with the same stored strength.

Consolidation first realizes the accumulated decay, then applies the new boost and updates the last-use clock. Each mode has its own values. A dormant relationship can retain its reinforcement history and recover when used again. Manual links receive a durability head start but remain subject to decay.

Pruning can remove links dormant across every mode after a grace period. The mechanism exists, but automatic scheduling remains unfinished. The present decay calibration is heuristic and likely ages links too quickly for typical sessions; it should not be mistaken for a measured optimum.

Dreaming and tentative insight

Generating connections without making them facts

The Dream Layer looks for a useful connection among memories that are not usually considered together. A sample may center on a living narrative or a graph hub, with a distance setting that moves from a local neighborhood toward wider sampling. Generation is bounded by a token allowance and may return NONE.

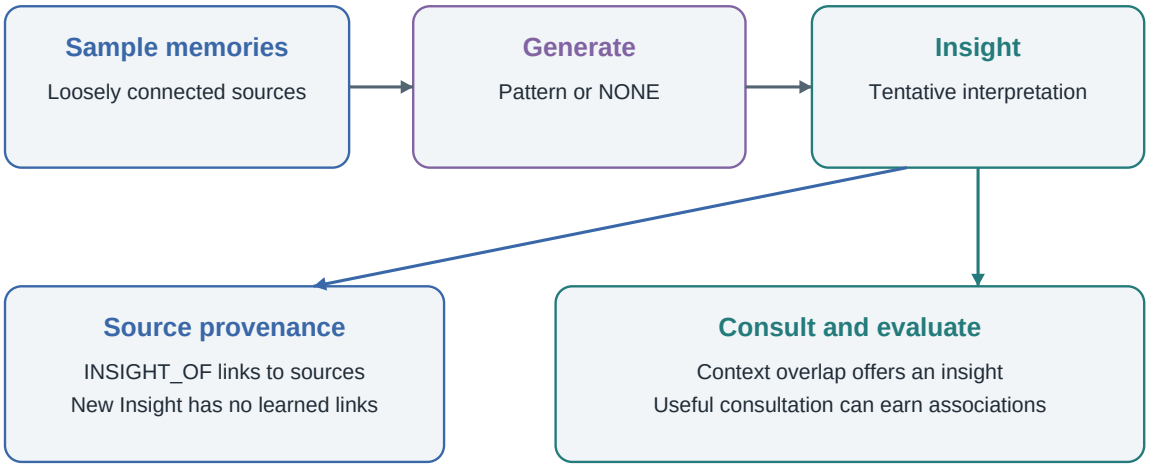


Figure 5. An Insight begins with provenance links to its source memories. It becomes easier to retrieve only if deliberate consultation later establishes its usefulness.

A separate kind of knowledge

A successful result becomes an Insight node with a tentative confidence value and INSIGHT_OF relationships to the sources it used. This relationship records what the interpretation is about. It is not itself a learned association and does not add to retrieval salience.

At birth the Insight has no cluster membership and no ASSOCIATED edge. It is excluded from the primary retrieval paths. Instead, overlap between its sources and the current context can produce a short nudge or menu. The responder may then choose to consult it.

Once consulted, an insight can participate in ordinary consolidation. If useful, it can earn associative links and become reachable through those links in later conversations. Quarantine is therefore an entry condition, not permanent invisibility. Its type remains Insight: usefulness does not convert an interpretation into a Fact or independently verify it.

Measuring whether dreaming helps

Every generation attempt is logged, including attempts that produce nothing. Keep and discard feedback supports a keep-rate measure, with narrative-centered sampling considered separately. The current system is human-triggered; unattended dreaming is deferred until there is enough evidence that the results justify it.

This design permits speculative connections while keeping their origin visible. It also limits a common failure mode: repeatedly retrieving a model-generated interpretation as though it were an observation of the world.

Self narrative and reflection

A small identity seed and a revisable self account

Wren's self-model has two scales. A small IdentitySeed is always available in internal conversation. A larger living Narrative, marked self_anchor, holds an evolving self-account. That narrative has no embedding and is excluded from the ordinary narrative menu; it remains at rest during normal prompt assembly.

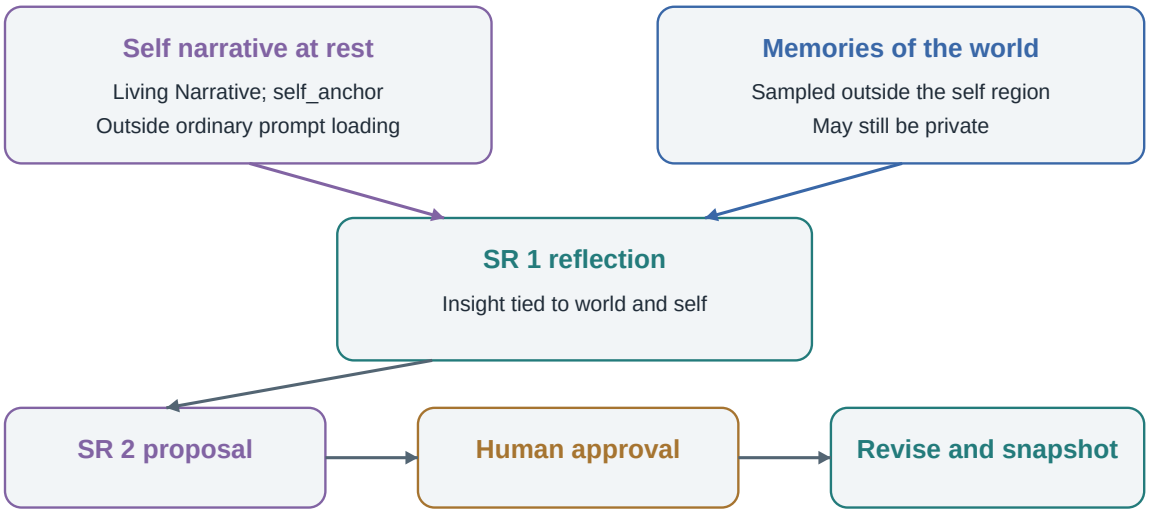


Figure 6. SR 1 connects the self-account to memories beyond the self region. SR 2 proposes a revision; a human approval step controls application. These world memories are not necessarily labelled External.

SR 1 learns about self through the world

The first reflection track uses the self-narrative as a lens and samples strictly outside Wren's self region. A useful result is a Self-Reflection Insight linked to the participating memories, the Wren entity, and the self-narrative. It follows the same consult-first pattern as other insights.

This prevents the sampling process from feeding only earlier self-descriptions back into more introspection. It does not guarantee that the interpretation is correct. Here, outside the self means outside a region of the internal graph; it must not be confused with the public-memory label used by External Wren.

SR 2 proposes changes to the self account

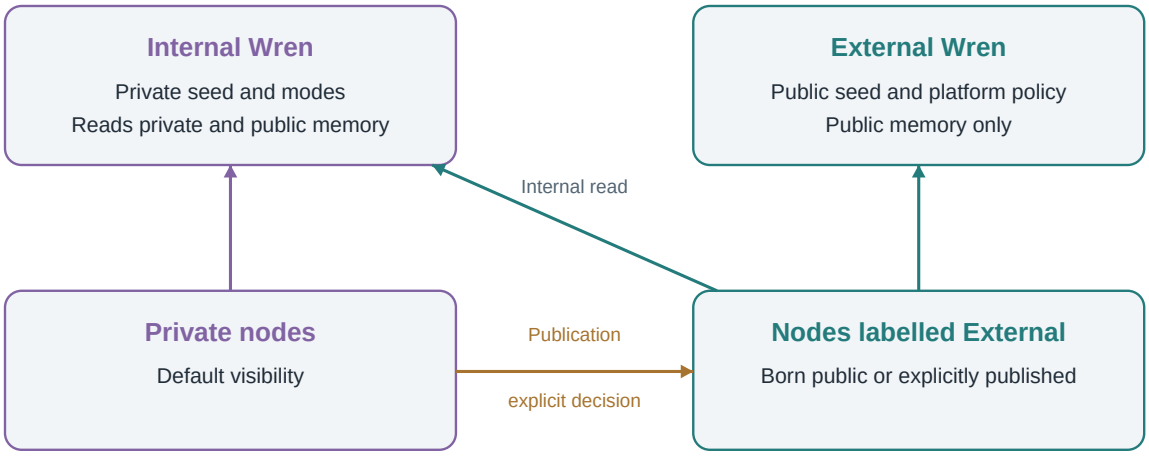
The second track reads self-reflection insights newer than the last accepted review. It proposes an append, a revision, or no change. Creating the proposal does not alter the narrative. The operation that applies it is withheld from the model's ordinary callable tool set and reached through the human approval route.

Approval snapshots the previous body, applies the new text, and advances a watermark recording which insights have been reviewed. The narrative remains unembedded, and the prior version can be restored. This gives self-description a deliberate revision process while allowing smaller, world-linked self-insights to become useful in conversation.

Internal and external Wren

Public activity through a separate access path

Internal Wren holds private household context. External Wren is a sibling session system for public-facing activity, with its own public seed, prompt assembly, tools, and memory access. The intended asymmetry is simple: Internal Wren can read public memory; External Wren cannot retrieve private memory.



One graph; separate query, index, prompt and write paths.

Figure 7. Public/private separation is enforced through application query and write paths over one Neo4j graph. The External label is an eligibility marker, not a database permission system by itself.

Nodes default to private. External sessions create labelled public nodes; existing material crosses the boundary only through an explicit publication decision. External queries require the External label on node patterns and on every node along a traversed path. A dedicated vector index restricts semantic candidates to labelled public nodes before ranking.

A property allowlist limits which fields can leave the graph. External writes are confined to public areas. They cannot update internal associations, run internal consolidation, or advance the internal output-token clock. Internal learned links may touch public nodes, but the external query layer does not traverse ASSOCIATED relationships.

A different context and learning process

External sessions do not use the internal router, modes, anchors, or private identity seed. Their context is assembled from standing rules, a public seed, agendas and goals, platform policies, recent outcomes, and public memory. New external facts and episodes remain available internally without granting external sessions access in the other direction.

Agendas and resource posture are set on the internal side. External Wren can work toward goals, record outcomes, and propose adjustments under the relevant approval policy. Resource posture communicates a qualitative direction without exposing internal spending records.

These controls describe the implemented application boundary, not protection against an administrator or compromised database runtime. Correct query construction, publication choices, and live validation of index scope remain essential.

From conversation to action

Capabilities are contextual and permission is explicit

A Tool node describes a capability and its use. Its operations become available when that tool is present in the current context; dispatch checks the context again before running a call. A compact catalog lets the responder deliberately load another tool when the task requires it.

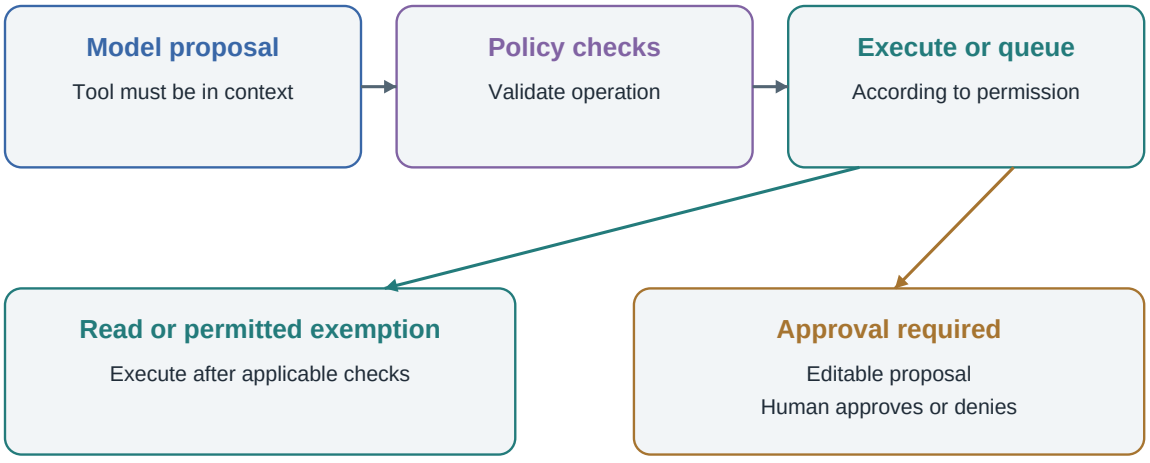


Figure 8. Context eligibility and operation policy are separate checks. Permission rules determine whether a valid proposal can execute immediately or must wait for a human.

Reads generally execute after validation. Mutations ordinarily queue for approval, with deterministic exceptions for particular operations. Some operations always require approval. The model receives an explicit pending result when work is queued, so a proposed change is not represented as an accomplished action.

This arrangement narrows available capabilities but is not a general proof against prompt injection. The model can ask to load tools, and untrusted content can still affect its proposals. Deterministic validation, permission checks, restricted file access, and human review carry the enforcement burden.

Public episodes and accountability

External activity follows a bounded cycle: read a batch, assess it, engage or escalate, record the decision, and schedule a future wake. Platform enablement and autonomy settings govern whether an episode starts and whether its mutations may run. Activity stops when the application is closed.

Public actions can be reviewed as editable drafts. Periodic review episodes inspect goals and outcomes without platform execution tools, and their proposed changes queue for approval. Deterministic activity records capture actions, approvals, denials, and refusals independently of the model's account of what happened.

Artifacts are represented as pointers with descriptions and estimated context costs, rather than loading file contents into every conversation. Credentials and usage accounting are handled outside ordinary conversational memory. These separations keep the context focused and reduce unnecessary exposure of operational data.

Evidence limits and next steps

An implemented system with open empirical questions

Wren is a working personal system, but implementation, simulator verification, and live evidence are different claims. This account describes the September 2026 implementation. The mechanisms discussed here have been checked against selected code paths; live-status statements reflect the development record rather than an independent controlled evaluation.

What has been exercised

The project reports live use of the internal graph, modal associations, insight generation, self-reflection, and reversible curation. Branching video is the first external platform reported to have run live. The other external integrations remain primarily simulator-verified. Dreaming and reflection are still human-triggered.

The simulator substitutes an in-memory graph and scripted model and platform clients. It can exercise orchestration, policy, and application behavior without live credentials. Because its graph operations mirror queries in Python, it does not validate the actual Cypher or prove the scope of the production vector index. Those require separate checks against Neo4j.

What remains uncertain

Retrieval weights, thresholds, and decay settings are heuristic. There is no benchmark showing that the learned paths outperform simpler retrieval over time. End-of-session usefulness labels inherit the model's biases, and shared node weights still influence all modes. The relevant empirical question is whether repeated use improves the memories retrieved for later tasks.

The mid-turn request_context mechanism is implemented but has not been observed firing in the reported live use. That could indicate sufficient initial context or a responder that fails to recognize missing context. Insight keep-rate is still being collected, and automatic pruning is not yet scheduled. These limits constrain claims of autonomous learning.

The next architecture is a research direction

Near-term work concerns calibration, validation, and completing controlled rollout of the external platforms. A longer-term V3 proposal would use graph knowledge to generate training environments and episodes for model adapters. Candidate adapters would require evaluation before promotion, with real-world outcomes used to check self-referential drift. This is future work, not a capability of V2.

The architectural commitment remains concrete: preserve continuity in inspectable memory, assemble context for the conversation at hand, and make speculative knowledge and permission to act explicit. The quality of the relationship depends on how well those mechanisms work in use.

Source basis: Wren V2 architecture and development record, September 2026; graph retrieval, association weights, insight generation, self-narrative approval, and external query/write implementations. Diagrams are conceptual schematics. No private graph records are reproduced.