# LCSM / Backboard

Durable state, provenance, authority, and inspectable change of mind for AI systems

Hayden Lindley · Glyphd Labs

2026-07-27

## Contents



# LCSM / Backboard

## Abstract

Long-running work with language models fails less often because the model cannot generate text than because the surrounding system cannot preserve what is currently true. Transcripts accumulate proposals, corrections, reversals, sources, and outputs, but they do not reliably distinguish what was said from what was accepted. A later model must reread history and reconstruct state, often blending obsolete instructions with current intent.

**Long-Context State Memory (LCSM)** is a model-agnostic runtime for durable canonical state, provenance, authority, journaled transition, and reconstruction. **Backboard** is the reference workbench that makes the protocol visible. Conversation is intake; bounded notes and canonical objects represent current recognized truth; immutable versions preserve change of mind; proposals are validated and gated before they move a canonical pointer; receipts explain why the current state exists.

The architecture is summarized by one chain of command:

> **The model proposes. The runtime validates. The policy gate authorizes. The journal records. The canon pointer moves. The user can inspect.**

LCSM is not an attempt to place an infinitely long transcript inside a model. It removes the obligation for the model to own memory. The model receives a scoped working set compiled from authoritative state and relevant evidence. The runtime retains the durable object.

## 1. The failure of transcript-as-memory

A transcript is useful evidence of interaction. It is not, by itself, a reliable state system.

Suppose a project contains the following sequence:

1. The user suggests a red interface.
2. The assistant proposes blue for accessibility.
3. The user says red is still preferred.
4. A later specification adopts a dark ink and violet identity.
5. A coding agent generates a blue component.
6. A design review rejects it.
7. A new agent opens the transcript and reads all seven events.

What is true now?

The transcript does not answer without interpretation. It contains every position, including superseded ones. Recency may help, but the latest message may itself be an assistant proposal, a failed output, or a question. Vector retrieval may find semantically related passages, but relevance is not authority. A summary may flatten the dispute and omit the reason for the decision.

The problem becomes more severe when:

- multiple models participate;
- conversations fork;
- files and tool results arrive asynchronously;
- decisions are corrected after implementation;
- personal memory and project memory overlap;
- a provider's chat is no longer available;
- old instructions remain semantically similar to current ones;
- a model confidently rephrases a proposal as a decision.

LCSM treats the transcript as an archive and source—not as the current mind.

## 2. Canonical state

Canonical state is the set of objects that the system presently recognizes for governed use. It is not necessarily metaphysical truth. It is the current, inspectable, authority-qualified record.

An LCSM object may be:

- a project constraint;

- an accepted decision;
- a source-backed fact;
- a user preference;
- an active plan;
- an unresolved question;
- a policy;
- a permission;
- an artifact reference;
- a dependency;
- a claim under dispute;
- a standing instruction;
- an uncertainty item.

Every object must make its epistemic and institutional condition visible. A source observation, a model inference, a user decision, and a verified test result are not interchangeable merely because they can be expressed in similar prose.

A minimal canonical object includes:

```
object_id
object_type
schema_version
current_version_ref
authority_basis
confidence_state
visibility
source_refs
dependency_refs
created_at
effective_from
effective_until
status
```

The exact schema can vary by domain. The invariant is that current state is represented directly rather than inferred from narrative.

## 3. Append-preserving correction

LCSM is append-preserving in effect. A correction does not silently overwrite the historical record. It creates a new version, supersession, invalidation, reversal, restoration, or adjudicated amendment.

A note version can be modeled as:

```
NoteVersion {
  note_id
  version_id
  parent_version_hash
  content
  shape
  sources[]
  actor
  authority
  timestamp
  receipt_ref
  version_hash
}
```

The current note points to one immutable version. A restore appends a new version that repeats or derives from an older state; it does not rewind the journal and erase the intervening history.

This design provides three forms of safety:

1. **Historical safety.** The system can explain what changed and when.
2. **Operational safety.** A failed candidate never destroys the last accepted state.
3. **Interpretive safety.** A later model receives the current pointer plus the relevant change history, not an undifferentiated transcript.

## 4. Conversation as intake

Conversation remains essential. It is the richest surface for ambiguity, reasoning, emotion, exploration, and correction. LCSM changes its role.

A message enters the archive exactly as captured. A classification process may identify:

- no durable effect;
- a candidate fact;
- a candidate preference;
- a proposed decision;
- a contradiction;
- a source;
- an instruction;
- an uncertainty item;
- a request to restore or supersede state.

The classifier does not mutate canon. It identifies the touched objects and prepares a working set. A model may draft a proposed change, but the proposal remains separate.

The reference loop is:

```
message
→ immutable archive record
→ classification
→ touched-object selection
→ working set
→ mutation proposal
→ old/candidate comparison
→ loss and contradiction checks
→ stakes gate
→ append immutable version
→ journal receipt
→ update current pointer
→ refresh projections and dependents
```

The design deliberately allows semantic judgment to be replaceable. A scripted classifier, local model, cloud model, or human can propose the mutation. The state machine and authority rules do not change.

## 5. The two-note commit

Backboard makes a proposed change visible as two bounded notes:

```
CURRENT NOTE                  CANDIDATE NOTE
what is accepted now          what would become accepted
```

```
sources and authority          proposed sources and authority
dependencies                   expected dependency effects
```

The comparison is not merely a text diff. It asks:

- What information is removed?
- Which source or authority changes?
- Does the candidate increase certainty without evidence?
- Does it convert a suggestion into a directive?
- Does it collapse a disputed state?
- Which dependent artifacts become stale?
- Is the requested change reversible?
- Does the proposal exceed the actor's authority?

This two-note form is one of the core interaction contracts. It converts “the assistant changed its mind” into an inspectable state transition.

## 6. Stakes and authority

Not every mutation requires the same ceremony. A system can classify stakes, for example:

- **S0:** ephemeral or presentation-only;
- **S1:** low-impact project state;
- **S2:** consequential plan, budget, permission, or accepted artifact;
- **S3:** high-impact identity, publication, external action, legal, medical, financial, or irreversible state.

The classification determines whether a change may be automatic, requires confirmation, or must hard-stop for a named authority.

Authority is not a sentence in a system prompt. It is runtime state. A model cannot gain authority by claiming it. A tool result cannot become canon because it arrived. A high-confidence source cannot automatically override the user. The policy gate evaluates actor, scope, object type, stakes, evidence, and standing permission.

## 7. Journal and receipts

Every consequential state transition emits a journal operation and a receipt.

A journal operation identifies:

```
operation_id
operation_type
target_object
prior_version
candidate_version
acting_actor
authority_basis
source_refs
policy_result
timestamp
correlation_id
```

A receipt adds the evidence and explanation necessary to inspect the transition:

```
receipt_id
request_ref
prior_state_ref
```

```
result_state_ref
checks[]
warnings[]
uncertainty
affected_dependents[]
rollback_route
acceptance_state
hash_chain_parent
receipt_hash
```

The journal is not merely an audit log written after the fact. It is the mechanism by which current state is reconstructed. A materialized board can be deleted and rebuilt from the journal. If deleting a projection loses truth, the projection has accidentally become a second authority.

## 8. Provenance walk

A user should be able to ask:

- Why does the project believe this?
- When did this become current?
- Who authorized it?
- Which source supports it?
- What did it replace?
- Which artifact was built from the old version?
- What becomes stale if this changes?
- Can it be restored?
- Is it fact, preference, policy, inference, or unresolved uncertainty?

LCSM answers by traversing object versions, source references, decisions, journal operations, and dependent artifacts.

This is more than transparency. It is a practical context compiler. When a model works on one object, the runtime can include only:

- the current canonical version;
- the relevant prior mutation;
- the authorizing rule;
- exact source spans;
- active dependencies;
- unresolved contradictions.

The model receives a smaller, cleaner, more authoritative context than a transcript dump.

## 9. Working sets and low model distance

Backboard projects canon into bounded notes on a board. The board is not the truth store; it is an operational view optimized for human and model access.

“Low model distance” means that the information required for a decision is already close to the active object in a typed working set. The model does not need to rediscover the current project identity, scan an entire chat history, or infer which version is accepted.

A working set may contain:

```
active objective
current canonical notes
touched sources
```

```
recent mutation trail
dependent artifacts
open uncertainty
applicable policy
requested output contract
```

The working set can be rendered to a person, serialized to a PromptCapsule, or placed on a FOVEA address surface. Its contents are derived from LCSM and disposable.

## 10. Artifacts and staleness

Canonical state is not isolated from making. An artifact declares the state versions on which it depends.

```
ArtifactManifest {
  artifact_id
  artifact_version
  input_refs[]
  canonical_note_versions[]
  tool_or_model_refs[]
  output_digest
  verification_refs[]
  acceptance_state
}
```

When a dependency moves, the artifact can become:

- current;
- possibly stale;
- definitely stale;
- unaffected;
- requiring regeneration;
- requiring human review.

This eliminates a common failure of AI projects: changing a decision without realizing that a specification, deployment, prompt, or design was built from the old value.

## 11. Multiple projections, one truth

LCSM can support many interfaces:

- a conversation;
- a board of bounded notes;
- a project timeline;
- a radial context selector;
- a FOVEA atlas;
- a mobile decision strip;
- a public proof page;
- an agent working set;
- a compliance report.

These are projections. They may cache or transform canonical state, but they may not become independent truth stores.

The rule is:

> A projection may improve reachability. It may not silently change authority.

This is the basis for the later PolyView discipline: many synchronized surfaces resolving to one governed state.

## 12. Reference implementation evidence

The Backboard v0 specification defines a complete deterministic loop over a realistic fixture corpus:

- immutable archive;
- classification;
- touched-note selection;
- working-set generation;
- mutation proposal;
- two-note comparison;
- loss check;
- stakes gate;
- immutable version append;
- hash-chained journal;
- board update;
- provenance walk;
- dependent artifact staleness;
- restore as append, not rewind.

The semantic judgment in the v0 demonstration may be scripted. The state machine, hash chain, gate, journal, and projection behavior are intended to be real production mechanics.

This distinction matters. The demonstration does not prove a universally correct memory classifier. It proves that semantic judgment can be contained inside a governed transition architecture.

## 13. Evaluation contract

LCSM should be compared with transcript-only memory, summary memory, and retrieval-over-history systems on long-running project tasks.

### Correctness

- accepted-constraint retention;
- superseded-state exclusion;
- source and authority accuracy;
- restore correctness;
- journal reconstruction;
- cross-project isolation;
- stale-dependency detection.

### Efficiency

- tokens required to resume;
- time to recover current truth;
- number of model calls;
- context bytes moved;
- user corrections caused by obsolete state.

**Human comprehension**

- can the user identify current truth?
- can the user distinguish their decisions from model proposals?
- can the user explain why a state is current?
- can they restore an earlier state without destroying history?
- can they see which artifacts are stale?

**Failure tests**

- broken hash chain;
- stale approval;
- missing source;
- authority spoof;
- cross-project retrieval;
- partial commit;
- crash between append and pointer movement;
- duplicate/idempotent operation;
- conflicting writers;
- corrupted local cache.

## 14. Security and privacy

A durable memory system can become dangerous if it confuses persistence with permission.

LCSM requires:

- project-scoped access;
- explicit personal/project/working/procedural planes;
- source visibility rules;
- revocable provider access;
- append-only evidence for consequential changes;
- export and deletion behavior;
- no automatic promotion of private chat to public registry;
- no whole-history transfer by default;
- single-writer or explicit conflict control;
- quarantine for corrupted or untrusted imported state.

An imported note does not become trusted because it fits the schema. Trust derives from source, authority, signature, project scope, policy, validation, and acceptance.

## 15. Limitations

LCSM does not solve semantic judgment. A classifier can still misunderstand the user, propose a poor note, or omit a relevant dependency. The architecture makes those errors visible and reversible rather than impossible.

Canonical state can also become over-structured. Some creative work benefits from ambiguity and contradiction. The runtime must permit unresolved, plural, or provisional state rather than forcing one answer.

The architecture creates storage and governance overhead. It requires version schemas, transactions, receipts, and projection invalidation. Those costs are justified for durable consequential work but may be excessive for disposable conversation.

Finally, “canonical” must not be mistaken for “objectively true.” LCSM records what the system recognizes under current evidence and authority. It must preserve disputed, inferred, simulated, and external states as distinct classes.

## 16. Conclusion

A model with a larger context window still does not become a trustworthy state system. It receives more history, but the burden of determining current truth remains inside probabilistic interpretation.

LCSM moves that burden into an inspectable runtime. The model can reason brilliantly without owning memory. Conversation can remain expressive without silently becoming canon. Changes of mind can be preserved without rewriting history.

Backboard makes the principle visible:

> Chat is the front of the assistant. Canon is the back.

The result is not a chatbot that remembers more. It is a state-maintaining intelligence environment in which memory, authority, evidence, and change can be inspected.

## Architecture illustration briefs

- **P02-F01 — Transcript versus canonical state**: Left lane shows a transcript containing conflicting statements. Right lane shows an archive feeding versioned canonical notes with one current pointer.
- **P02-F02 — The LCSM chain of command**: LLM proposes → runtime validates → policy authorizes → journal records → canon pointer moves → projections refresh, with human inspection at every consequential gate.
- **P02-F03 — Two-note commit**: Side-by-side current and candidate notes, including sources, authority, information loss, dependency effects, and acceptance controls.
- **P02-F04 — Projection reconstruction invariant**: One journal fans out to chat, board, FOVEA atlas, mobile strip, agent capsule, and public proof page; deleting any projection leaves canon intact.

## Source register

- **P02-S01** — LCSM Workbench E2E Specification v0 (2026-07-04). `LCSM-WORKBENCH-E2E-SPEC.v0.md`
- **P02-S02** — Glyph Desk Master Specification v1 (2026-07-12). `GLYPH-DESK-MASTER-SPEC.v1.md`
- **P02-S03** — Continuity Core Design Review 2 (2026-07-21). `CONTINUITY-CORE-DESIGN-REVIEW2.md`
- **P02-S04** — Glyphd V1 Canon Recovery and Gap Audit (2026-07-18). `ZEKE-V1-CANON-RECOVERY-AND-GAP-AUDIT-v0.1.md`
- **P02-S05** — MCE-1 System Contract (2026-07-25). `MCE-1-SYSTEM-CONTRACT-v0.1.md`

## Public-disclosure and IP boundary

This edition publishes the state architecture, transition doctrine, and public-safe reference loop. It excludes private user data, confidential classifier prompts, unpublished policy thresholds, security-sensitive storage details, and patent-claim drafting. The term “canonical” describes governed recognized state, not an assertion of universal truth.