Hidden-State Memory Research Report

Date: 2026-04-25

Executive Summary

This report summarizes the memory research path that led from latent-vector memory, through LoRA-reader and direct-latent conditioning attempts, into the current hidden-state memory architecture. It is written as a standalone account: no prior knowledge of the repository or earlier project notes is assumed.

The central research question was whether long-horizon memory could be stored, organized, and retrieved as model-native or latent vectors rather than as plain text summaries. The answer we reached is mixed but useful:

- Vector-first memory is a viable substrate for retrieval and organization.
- Hierarchical abstractions help retrieval when they guide descent toward relevant episodes.
- Frozen-model hidden states contain useful semantic signal, especially in later layers, but raw cosine similarity does not separate the distinctions memory needs.
- Slot-governed and lattice-governed abstraction prevent dangerous structural merges, especially for revisions and default/override cases.
- The abstractions we currently build are not load-bearing as standalone generation inputs. They organize retrieval, but when concrete episodes are removed, generation accuracy collapses.

The practical conclusion is that this path should not continue by tuning thresholds, layers, or raw cosine policies. The architecture taught us something clear: abstractions are useful as retrieval structure, but the current abstract payloads are not sufficient state objects for generation.

What We Were Trying To Build

The original goal was a memory system whose primary representation was not language. Text should be a rendering layer used when a language model needs to consume retrieved memories, not the memory substrate itself.

The desired system shape was:

1. Store concrete episodes.
2. Encode them into vectors.
3. Consolidate similar episodes into abstractions.
4. Organize episodes and abstractions into a hierarchy.
5. Retrieve through that hierarchy.
6. Render only the retrieved state needed for downstream reasoning or generation.

This was motivated by a long-horizon memory problem: raw episodes are detailed but grow without bound; abstractions compress repeated structure and should make old memory cheaper while preserving useful regularities.

The First Vector-Primary System: Symbolic Latent Memory

The first working vector-primary implementation used symbolic slots as the source of truth. A structured world state, such as a fact, trait, or override, was converted into a latent vector by assigning deterministic unit vectors to symbols and summing them.

For example, a fact about Atlas's deadline could be represented as slots:

```
kind = fact
entity = atlas
fact_kind = deadline
fact_value = date:2025-06-14
source = src_standup_05
```

Each slot-value pair received a deterministic vector. The memory vector was the normalized sum of those vectors. Because vectors were unit-normalized, inner product was cosine similarity. This let the system retrieve memories that shared structural components such as the same entity, fact kind, trait name, or current freshness marker.

In this branch:

- episodes were vector objects with symbolic metadata;
- abstracts were normalized centroids of child vectors;
- retrieval used vector search and hierarchical descent;
- text appeared only at the final rendering step.

This system was deliberately synthetic. It did not prove that frozen language model hidden states could carry the structure. It proved that the overall architecture - vector episodes, vector abstracts, hierarchy, retrieval, and rendering - could work when the geometry was clean.

Earlier Generation Attempts: Text Briefs, Direct Latent Prefixes, And LoRA

The project then tested whether a language model could use the retrieved memory. Several approaches preceded the hidden-state branch.

The first approach rendered retrieved memories as text briefs for a frozen LLM. This preserved compatibility with ordinary prompting, but it collapsed much of the typed structure that made the memory useful. Fact, trait, override, revision, parent-child, and provenance distinctions became prose. The model could read the prose, but the retrieval hierarchy's structural advantage did not reliably transfer into generated answers.

The second approach passed retrieved latent vectors through a lightweight projector as direct prefix embeddings. Training loss moved in the right direction, which suggested the model attended to the prefix. But the frozen model had not learned how to interpret those projected vectors as structured memory content. The interface was too weak.

The LoRA-reader plan was the next proposed bridge. It kept memory external, vector-first, and hierarchical, but added a trainable reader inside the language model. In that design, LoRA was not the memory substrate. It was a translation layer, analogous to a multimodal adapter: retrieved memory vectors would be projected into the model, and LoRA weights would teach the model how to consume them.

That line of work clarified an important distinction:

- memory organization can succeed outside the generator;
- generation requires a strong enough memory-consumption interface.

The hidden-state path inherited that lesson.

Hidden-State Based Encoding

The hidden-state branch replaced synthetic symbolic vectors with dense vectors taken from a frozen language model. Each episode was wrapped in a deterministic prompt, passed through the model, pooled from a selected hidden layer, and L2-normalized at write time.

The intended encoder contract was explicit:

- fixed model id;
- fixed layer index;
- fixed pooling rule;
- fixed normalization rule;
- versioned memory format.

The implementation uses normalized hidden-state vectors for episode search and abstract routing. Because vectors are normalized, dot product and cosine similarity are equivalent.

This was a major shift:

- The old symbolic-latent system had clean geometry because slots generated the vectors.
- The hidden-state system asked whether a frozen model's internal activations already contained enough useful structure to support memory consolidation and retrieval.

The answer was: partially. Hidden states carry semantic signal, but not in a form raw cosine can use safely.

From Centroid Abstractions To Cosine-Governed Clusters

The early latent system treated an abstract largely as the centroid of its children. That was reasonable in the synthetic latent space: if child vectors were generated from clean symbolic slots, their centroid was a meaningful summary direction.

The hidden-state system forced a change. A centroid over frozen hidden states was not reliable enough to be treated as a complete semantic object. We moved toward a more modest interpretation:

- pairwise cosine forms local clusters;
- abstracts become organizational nodes over child memories;
- abstracts carry routing/schema vectors and metadata;
- structure, provenance, slot identity, and revision relations remain explicit.

In other words, cosine similarity remained useful, but only as a local grouping tool. The abstract node stopped being "the meaning" of its children and became a retrieval and compression object with explicit structural metadata.

This distinction matters. The project did not show that cosine centroids can be trusted as general-purpose memories. It showed that cosine can help organize memories once the system constrains the comparison to a well-posed partition.

The Raw Cosine Failure

The most important negative result came from the P1/P2 layer sweep on `Qwen/Qwen2.5-7B-Instruct`.

The benchmark compared two kinds of pairs:

- P1 twins: memories that should cluster because they are paraphrases or same-meaning variants.
- P2 confusables: memories that are topically similar but must not merge, such as stale versus current revisions, negated values, default versus override, same entity with different fact kind, or same fact value on different entities.

Across 29 hidden-state layers and 2610 sweep configurations:

- production raw-cosine joint-pass configs: `0`;
- constrained slot-governed joint-pass configs: `214`;
- every layer was distributionally inverted;
- P1 mean cosine: `0.597`;
- P2 mean cosine: `0.821`;
- P2 max cosine: `1.000`.

The result was not that hidden states contain no useful information. Late layers made P1 twins much closer. The problem was that P2 confusables rose with them. Depth amplified topical and semantic relatedness without separating the propositional distinctions memory needs.

The key reading was:

```
The stack carries signal. Cosine cannot carve it safely by itself.
```

This closed the "maybe a different layer rescues raw cosine" path.

Slot-Governed Consolidation

After the cosine failure, slot/value/temporal structure became the substrate for safe consolidation. Cosine was allowed to operate only inside typed lanes where comparison was

well-posed.

The Stage B policy surface distinguished three policies:

- `production` : raw or legacy production consolidation, retained as the negative control.
- `slot_partition` : partition by slot identity before cosine clustering.
- `slot_value_partition` : partition by slot identity and value/temporal fields before cosine clustering.

This was not a threshold tweak. It changed the question from:

```
Are these two memories close in hidden-state space?
```

to:

```
Are these two memories about the same kind of thing, in the same structural
lane, and then close enough in hidden-state space?
```

That prevented merges that raw cosine repeatedly allowed.

Hierarchical Organization: Episodes, Abstracts, And The Forest

The hierarchy work generalized the memory store beyond a fixed two-level shape. The system distinguished:

- leaf nodes: concrete episodes;
- interior nodes: abstracts;
- root nodes: abstracts with no parent.

Retrieval began from the root frontier, descended through promising abstract children, and kept global episodic retrieval as a backstop. This produced a ragged forest rather than a single rigid tree: some branches could grow deeper than others.

The hierarchy also separated direct evidence from transitive evidence. A high-level abstract could point to many descendants, but it should not be treated as if it directly asserted every child detail. That distinction was important for avoiding overconfident generation from broad abstractions.

The strongest retrieval results came from mixed retrieval:

- abstract-guided descent supplied structure;
- episode retrieval supplied concrete evidence;
- global episodic backstop protected against abstract routing failures.

This is a recurring theme in the project: abstractions help retrieval most when they route toward episodes, not when they replace episodes.

The Referential Lattice

The most important conceptual advance was the referential abstraction lattice.

The core idea is that abstraction should not be "cluster all similar content." It should be:

1. partition memories by what they are about;
2. cluster within each partition by content similarity.

An episode is treated as a claim with:

- a referent: the typed thing the memory is about;
- content: what it says about that referent.

For example:

```
(entity = atlas, attribute = deadline)
(entity = atlas, attribute = tone)
(entity = beacon, attribute = deadline)
```

The lattice arises by projecting referents at different granularities:

- Level 0: raw episodes.
- Level 1: identity projection, such as `(atlas, deadline)`.
- Level 2: coarser projections, such as `(*, deadline)` or `(atlas, *)`.
- Higher levels: broader projections if the slot vocabulary supports them.

This lets cross-cutting categories emerge without forcing one vector to encode all abstraction axes at once. An entity can participate in multiple abstraction paths because it appears in multiple projected cells.

For example:

- apples can participate in a `fruit` abstraction through a category axis;
- apples can participate in a `round objects` abstraction through a shape axis;

- those are different lattice paths, not one ambiguous content cluster.

The lattice made abstractions useful in retrieval because it narrowed each cosine comparison to a well-posed local job. Cosine no longer had to distinguish "Atlas deadline" from "Atlas tone" in a global embedding space. The lattice first chose the relevant referential lane, then cosine compared content inside that lane.

The lattice also doubled as a compression schedule:

- recent memories keep raw episodes;
- older memories can retain level-1 abstracts;
- very old memories can retain higher-level abstracts;
- lower levels can be evicted once a higher-level parent is trusted.

This was the long-horizon memory mechanism: detail fades, but regularities survive as structured abstractions.

Relational Revision Memory

Mutable facts and overrides required more than clustering. The system needed to represent revision as a relation, not as a merge.

Stage C added explicit revision edges:

- `superseded_by`;
- `reverted_by`;
- `exception_ended`;
- `revision_edges`;
- `temporal_status`.

The writer grouped mutable slots generically:

- facts by `(kind, entity, fact_kind)`;
- overrides by `(kind, entity, task_type, trait_name)`.

When a later value changed a slot, the previous value was marked superseded. When a later value returned to an earlier value, the intervening exception was marked ended. These relations were also propagated into abstract metadata.

The Stage C scripted probe passed:

- strict pass: `true`;

- checkpoints: `12`;
- equivalence failures: `0`;
- edge count: `4`;
- orphan edge count: `0`.

This is one of the clearest engineering wins: revision handling should be a stored relation, not a side effect of whichever retrieved item happens to be newest.

Retrieval Benefits We Demonstrated

The retrieval story has several positive results.

First, the synthetic latent system showed that vector-first hierarchy can work when the geometry is clean. Hierarchical `mixed` retrieval beat flat or episode-only variants, and depth-sensitive synthetic tests showed recursive hierarchy succeeding where forced two-tier hierarchy failed.

Representative archived findings:

- recursive latent success around `0.97`;
- recursive depth-sensitive abstract and mixed success: `1.0000`;
- forced two-tier depth-sensitive abstract and mixed success: `0.0000`;
- tool-mediated LLM confirmation:
  - `mixed` overall exact: `0.9500`;
  - `mixed_flat` overall exact: `0.8750`;
  - `episodes_only` overall exact: `0.3000`;
  - `abstract_only` overall exact: `0.0000`.

Second, the hidden-state layer sweep showed that constrained policies can find valid clustering configurations where production raw cosine cannot. Out of 2610 sweep configurations, `214` joint-pass configurations existed, all under constrained policies.

Third, Stage B P4 showed that slot-governed consolidation preserves important structure:

- candidate `slot_value_partition` revision-loss events: `0`;
- candidate default/override merge events: `0`;
- production revision/default-override failures appeared under the negative control.

Fourth, Stage C showed that explicit revision edges can reproduce current-slot resolution on scripted hidden-state chat scenarios without orphan relations.

Taken together, these results support a bounded claim:

```
The abstraction hierarchy improves retrieval organization and structural
preservation, especially when paired with concrete episodic evidence.
```

They do not support the stronger claim that current abstracts are sufficient for generation.

Generation Results: Abstractions Were Not Load-Bearing

The generation and answer-accuracy results were the limiting factor.

In the initial Stage B P3 gate, both `slot_value_partition` and `production` showed the same answer-accuracy lift over `episodes_only`:

- candidate answer-accuracy delta: `0.5833333333333334`;
- production answer-accuracy delta: `0.5833333333333334`;
- candidate retrieval-precision delta: `0.0`;
- production retrieval-precision delta: `0.016666666666666663`.

Because the negative-control production policy matched the candidate, the lift could not be attributed to slot-governed abstraction.

The abstract-stress follow-up removed concrete episode retrieval:

- `top_k_episodes = 0`;
- `top_k_abstracts = 3`;
- candidate answer-accuracy delta: `0.0`;
- production answer-accuracy delta: `0.0`;
- both candidate and production retrieved abstracts on every checkpoint;
- `episodes_only` and no-retrieval controls achieved `0.0` answer accuracy.

This was decisive. Abstracts were being retrieved and included, but they did not carry enough explicit state for the controller to answer correctly without episodes.

The same pattern appeared earlier in the hierarchy work: `abstract_only` scored `0.0000` while `mixed` scored strongly. The hierarchy helped when abstracts guided retrieval toward concrete memories. It did not help when abstracts were the only evidence.

What The Abstracts Currently Are

It is important to be precise about what the current abstracts are and are not.

They are useful as:

- clustering records;
- routing objects;
- compression nodes;
- child/provenance containers;
- structural guards against invalid merges;
- carriers of aggregate revision metadata.

They are not yet useful as:

- complete semantic memories;
- standalone state packets;
- reliable generation context;
- sufficient replacements for concrete episodes.

That does not make them a failure. It tells us their current role. They are index and organization machinery, not cognitive summaries that a frozen model can reliably reason from.

Why Larger Episode Volumes Might Help, But Not In The Simple Way

Larger volumes of episodes would populate more lattice cells. That would make more abstractions statistically meaningful and create richer long-horizon compression opportunities.

However, more data alone does not fix the frozen representation's content similarity. The layer sweep showed that confusables remain close across the stack. More data can populate the referential lattice, but it does not teach cosine to separate negation, revision, default/override, or fact-kind distinctions unless the representation or partitioning changes.

The realistic benefit of larger data is:

- richer referential coverage;
- more stable abstraction cells;
- more useful hierarchy and compression;
- better tests of long-horizon retrieval.

The unrealistic expectation is:

- that raw hidden-state cosine will become safe merely because the corpus is larger.

What We Learned

The main lessons are:

1. Vector-first memory is viable for retrieval architecture.
2. Hidden-state vectors can serve as memory encodings, but not as an unguided consolidation substrate.
3. Cosine similarity is useful only after the system has constrained the comparison to a well-posed referential lane.
4. Abstraction should be modeled as referential partition plus within-partition similarity, not global content clustering.
5. Hierarchy helps retrieval when it routes toward concrete evidence.
6. Revision is a relation and should be stored as such.
7. Current abstracts preserve structure but do not carry enough explicit state to support generation by themselves.
8. The bottleneck has moved from memory organization to memory consumption.

Final Assessment

This research path produced a useful architecture and a useful negative result.

The positive result is that hierarchical, lattice-governed memory can organize retrieval better than flat retrieval and can preserve distinctions that raw cosine would collapse. The retrieval substrate is real.

The negative result is that the current abstractions are not load-bearing for generation. When the generator must rely on abstracts alone, answer accuracy does not improve. When production negative controls match candidate gains, the gain cannot be credited to the abstraction method.

The next productive research fork would not be another threshold sweep or layer search. It would need one of two deeper changes:

- richer abstract payloads that explicitly preserve answerable state; or
- a trained reader/compressor that learns how to consume abstract memory as a generation interface.

For the current path, the clean conclusion is:

```
Use abstractions as retrieval structure and compression structure.
Do not treat the current abstracts as sufficient generation memory.
```