Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

HierMem: Context Curation Over Context Scaling — Hierarchical Memory with Invariant Constraint Placement for Long-Horizon LLM Conversations

Yash Doke

yashdoke215@gmail.com

Progressive Education Society's Modern College of Engineering https://orcid.org/0009-0001-3105-8227

Research Article

Keywords: Large Language Models, Hierarchical Memory, Context Management, Retrieval-Augmented Generation (RAG), Long-Horizon Dialogue, Constraint Preservation, Natural Language Processing

Posted Date: June 17th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10055780/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

HierMem: Constraint-Preserving Hierarchical Context Management for Long-Horizon LLM Conversations

Yash Doke
Progressive Education Society's Modern College of Engineering
yashdoke215@gmail.com

May 2026

Abstract

The industry assumes that larger context windows are the solution to long-horizon conversational AI. This assumption is fundamentally flawed. Flooding a Large Language Model (LLM) with raw historical turns actively degrades reasoning through the "lost in the middle" effect, dilutes critical user constraints, and incurs massive compute costs. LLMs do not need to process every past interaction; they require only highly relevant information, curated intelligently like human recall.

I present HierMem, an OS-inspired systems architecture that explicitly challenges the full-history paradigm. HierMem replaces brute-force context dumping with a four-level hierarchical memory (L0–L3), an invariant non-evictable constraint zone, and a lightweight stateless curator model. By separating memory orchestration from response generation, constraint salience is guaranteed by placement policy, while topical relevance is handled by precision routing.

Evaluated on 15 synthetic benchmark conversations (50 turns each) across software engineering, finance, legal, and healthcare domains, HierMem achieves a 4.7x architectural compression ratio. In the finalized Architecture C run (Qwen2.5-14B main model), HierMem obtains the best aggregate judge score (8.461) and a 93.3% constraint survival rate. Relative to Raw LLM, HierMem improves quality by +1.553 judge points while reducing average session compute cost by 36.9%. This work demonstrates that structured curation, rather than context scaling, is the optimal path for reliable long-horizon assistants.

1 Introduction

Modern LLM assistants are no longer used only for short question answering. They are increasingly deployed in sessions that last dozens of turns and involve evolving context, persistent formatting rules, domain constraints, and user-specific preferences.

To handle these long horizons, the prevailing industry trend is to scale context windows to hundreds of thousands of tokens. This creates a recurring systems-level tension. If all prior history is passed forward, inference costs grow non-linearly and the model's attention becomes diffuse—causing it to forget strict, policy-critical constraints hidden in the noise. Conversely, if naive retrieval (RAG) is used aggressively, prompt size drops, but there is no guarantee that the highest-priority constraints are retrieved and positioned where the model attends most. Summary-based compression improves efficiency but suffers from semantic drift, especially when strict wording must remain unchanged for many turns.

The core thesis of this work is that long-horizon reliability is a *context management problem*, not a model-size or context-window problem. Larger context windows can delay failures, but they do

not eliminate positional salience effects or remove the economic pressure of repeated long prompts. LLMs do not need to see everything; they only need what is relevant, curated intelligently.

Inspired by OS-style memory hierarchy ideas and human recall mechanisms, I design HierMem to explicitly decompose memory responsibilities and enforce constraint visibility as a structural invariant.

Why this architecture is needed. A single large model can often answer based on parametric knowledge alone for many requests, but multi-turn user-specific constraints are conversation-local obligations. Retrieval systems can recover relevant history, yet retrieval relevance does not imply policy salience. Therefore, I separate routing and memory maintenance (using a small curator model) from high-quality generation (the main model). I force constraint text into a protected, non-evictable prompt zone each turn, allowing the system to aggressively prune up to 80% of the raw conversation history without degrading compliance.

Contributions.

- I challenge the full-history paradigm by formulating long-horizon assistant behavior as a tri-objective optimization over quality, constraint survival, and compute cost.
- I introduce a constraint-first hierarchical memory runtime with explicit L0–L3 separation, adaptive passthrough, and bounded assembly.
- I implement a dual-model pipeline in which a smaller curator/summarizer controls memory operations, yielding a 4.7x architectural compression ratio and reducing expensive main-model context usage by 36.9%.
- I report complete evaluation over 15 datasets across four domains, establishing that smart curation outperforms full-history prompting by +1.553 judge points.

2 Related Work

Retrieval-augmented generation. RAG combines parametric and non-parametric memory to improve knowledge-intensive generation [1]. Subsequent systems such as ATLAS and RETRO advance retrieval quality and scaling behavior [2, 3]. However, these methods primarily target factual grounding, while our focus is persistent user-constraint retention under long-horizon dialogue.

The Illusion of Infinite Context. The prevailing trend of scaling context windows—exemplified by GPT-4's 128k tokens [7], Claude 3's 200k tokens [8], and Gemini 1.5's million-token capacity [9]—relies on the assumption that models can effectively reason over massive, unstructured history. However, the "Lost in the Middle" phenomenon demonstrates that relevant information placed in middle positions of long prompts is severely underutilized [4]. Furthermore, sparse-attention (Longformer, BigBird) [10, 11] and positional bias studies (ALiBi) [12] prove that while models can physically *process* long sequences, they struggle to maintain strict instruction salience across multi-turn conversational updates. This directly motivates HierMem's approach: rather than expanding the context window to absorb noise, we aggressively prune the noise and enforce strict zone ordering to guarantee constraint salience.

Memory systems for LLMs. MemGPT proposes an OS-inspired memory management view for extending practical context handling [5]. HierMem shares this systems framing but differs in two key ways: (i) explicit non-evictable constraint injection every turn, and (ii) role separation with a smaller curator model for memory control.

Adaptive retrieval policies. Self-RAG introduces retrieve/generate/critique control via training-time reflection tokens [6]. In contrast, HierMem adopts modular orchestration without requiring end-to-end retriever-policy fine-tuning, prioritizing practical deployability in mixed-provider runtime environments.

Evaluation with LLM judges. MT-Bench and related work demonstrate that structured LLM-as-judge protocols can provide scalable model comparison when prompt templates and score aggregation are controlled [13]. I follow this paradigm with fixed checkpoint scoring and aggregate statistics.

Judge model and scoring configuration. Checkpoint scoring is performed with Gemini 3.1 Pro in Google AI Studio. I use this model because the larger context capacity helps preserve the rubric, checkpoint history, and active constraints in a single judging pass, which reduces truncation artifacts during evaluation. This is part of the benchmark configuration rather than a training-time choice.

Why the judge matters. The judge is not a cosmetic detail. In a long-horizon benchmark, the scoring model determines whether the paper measures actual constraint survival or only surface plausibility. I therefore keep the judge fixed across systems and checkpoints, and I treat the rubric as part of the method rather than as an invisible evaluation convenience. The benchmark prompt is designed to reward three things only: task completion quality, explicit constraint adherence, and conversational coherence under accumulated state.

3 Problem Formulation

The industry currently treats long-horizon dialogue as a token-capacity problem. I argue it is fundamentally a curation and scheduling problem. This section formalizes the design target for long-horizon assistants as a constrained multi-objective optimization problem. The intent is to make clear why naive full-history prompting (the current industry standard) is actively harmful to reasoning, and why basic retrieval and summary pipelines fail on policy retention.

Let the conversation state at turn t be:

$$\mathcal{H}_t = \{(u_1, a_1), \ldots, (u_t, a_t)\}, \qquad \mathcal{C}_t = \{c_1, \ldots, c_m\}, \tag{1}$$

where u_i is the user message at turn i, a_i is the assistant response at turn i, $\mathcal{H}_t$ is the full dialogue history up to turn t, and $\mathcal{C}_t$ is the active constraint set at turn t. A context manager constructs prompt P_t under token budget B:

$$P_t = [\text{System},\ \mathcal{C}_t,\ R_t,\ L_t,\ u_t], \qquad |P_t| \leq B, \tag{2}$$

where R_t is retrieved long-horizon context selected from hierarchical memory (L1/L2/L3) and L_t is the recent local window (most recent turns).

I evaluate each strategy along three deployment-critical objectives:

$$Q_t : \text{response quality}, \tag{3}$$
$$S_t : \text{constraint survival/adherence}, \tag{4}$$
$$C_t : \text{compute cost per turn}. \tag{5}$$

The target is high Q_t and S_t with low C_t:

$$\max(Q_t, S_t), \qquad \min C_t. \tag{6}$$

For scalar comparison, I use:

$$U_t = \alpha Q_t + \beta S_t - \gamma C_t, \qquad \alpha, \beta, \gamma > 0. \tag{7}$$

Here α, β, γ are task-priority weights: larger α emphasizes output quality, larger β emphasizes strict rule retention, and larger γ emphasizes efficiency pressure. They are not learned parameters in this work; they are analysis weights that make trade-offs explicit.

Most baselines optimize only a subset of this objective. Full-history prompting tends to preserve S_t early but worsens C_t rapidly; retrieval-only systems reduce C_t but can lose S_t when constraints are not foregrounded; summary-heavy methods reduce cost but can erode both Q_t and S_t via semantic drift. HierMem is designed to couple all three objectives through explicit context zoning and hierarchical memory operations.

4 System Design and Runtime

4.1 Architecture Overview

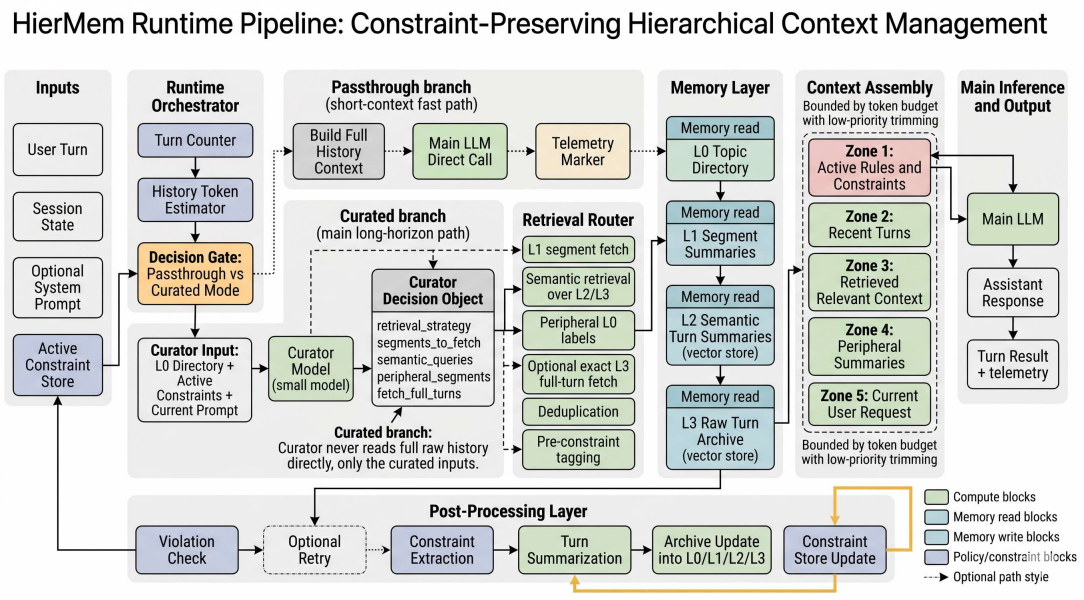


Figure 1: HierMem runtime pipeline with adaptive passthrough, curator-driven retrieval, and five-zone context assembly.

I design the runtime as a separation-of-responsibilities pipeline: the curator model controls memory movement and retrieval decisions, while the main model is used primarily for final response generation. This division keeps expensive inference focused on high-value decoding and moves lower-cost orchestration steps to a smaller model.

At each turn, HierMem follows the actual implementation path:

1. **Passthrough check**: if the conversation still fits under the adaptive threshold, the pipeline sends the full running history directly to the main model.
2. **Curator decision**: otherwise, the curator reads only the L0 directory plus the active constraint store and returns retrieval fields.
3. **Retrieval routing**: the router executes L1 fetches, semantic L2/L3 queries, optional peripheral L0 labels, and optional full-turn L3 fetches.
4. **Pre-constraint tagging**: retrieved chunks that predate active rules are marked so old patterns do not override current constraints.
5. **Context assembly**: the assembler builds the prompt in zone order with constraints first, recent turns next, retrieved context after that, peripheral summaries lower-priority, and the current request at the end.
6. **Main inference**: the main model generates the assistant response from the assembled prompt.
7. **Post-processing**: the pipeline checks violations, optionally retries, extracts new constraints, summarizes the turn, and updates L0/L1/L2/L3 memory.

This ordering is deliberate: retrieval is decided before assembly, assembly is finalized before generation, and post-processing closes the loop by turning response outcomes into memory updates. The passthrough branch is not a separate system; it is simply the short-context fast path inside the same pipeline.

4.2 L0–L3 Hierarchical Memory

- **L0 Topic Directory**: compact segment index (segment id + turn range + label). This is what the curator reads first to decide where to search.
- **L1 Segment Summaries**: short bullet summaries per segment used for structured fetch when a segment is selected from L0.
- **L2 Semantic Layer**: one summary record per turn stored in Chroma metadata level=L2 and queried semantically when the curator asks for topical recall.
- **L3 Raw Archive**: verbatim turn text (Chroma metadata level=L3) used when exact content recovery is required (for example, code or strict wording).

In the implemented runtime, only compact structure is always exposed to the curator (L0 plus active constraints). Deeper levels are fetched on demand: L1 for concise segment detail, L2 for semantic relevance, and L3 for exact recall. This design avoids repeatedly injecting full raw history while preserving recoverability for high-precision turns. Segment creation and controlled merging keep directory growth bounded.

4.3 Adaptive Passthrough and Curated Mode

Short conversations do not always require heavy orchestration. HierMem therefore uses adaptive passthrough: when estimated history is below threshold, full-history mode is used to avoid premature compression. Once history exceeds threshold, the pipeline switches to curated retrieval and bounded assembly. This hybrid strategy avoids both early information loss and late context explosion.

4.4 Constraint-First Zone Policy

Assembler zones are ordered to enforce salience:

1. Invariant constraints (non-evictable)
2. Recent turns (always included)
3. Curated relevant context
4. Peripheral summaries
5. Current user request

When over budget, low-priority zones are trimmed first. Constraint and current-request zones are preserved. This is the primary mechanism behind sustained adherence under long horizons.

4.5 Main LLM Call Path and Runtime Environment

Main inference and support calls are executed through a multi-provider API client abstraction (Ollama/OpenAI/Groq/Google/Anthropic). The benchmark runtime does not use an MCP inference path. This distinction is important for reproducibility and cost accounting, since call budgets, retries, and token pacing occur at provider-client level.

4.6 Why a Smaller Curator Model

A large model used for both generation and memory control can become cost-heavy due to repeated auxiliary calls. HierMem delegates routing, extraction, and summarization to a smaller curator/summarizer model, reserving large-model compute for final response quality. This increases the number of LLM calls, but most additional calls are low-token and executed on a cheaper/smaller model, so aggregate compute burden remains lower than repeatedly sending large contexts to the main generator. Empirically, this lowers aggregate compute cost while maintaining strong quality and survival metrics.

5 Experimental Setup

This section defines the evaluation protocol used to test whether HierMem improves long-horizon behavior under realistic multi-domain workload conditions. The setup is designed to answer a practical deployment question: can a constraint-aware memory architecture improve response quality and adherence while lowering effective compute burden over 50-turn conversations. To make this comparison valid, all systems are run under the same turn schedule, checkpoint schedule, and scoring process.

The benchmark corpus is synthetic and intentionally controlled. I use synthetic dialogues here because the goal is not to model natural conversation frequency, but to isolate memory failure

modes under known constraint lifecycles, long horizons, and repeated policy checks. This makes the comparison suitable for architectural evaluation, ablation design, and release reproducibility.

5.1 Datasets and Protocol

I evaluate 15 conversations of 50 turns each (750 turns total) spanning:

- Software engineering (6 datasets)
- Finance (4 datasets)
- Legal (3 datasets)
- Healthcare (2 datasets)

The domain-count imbalance reflects available synthetic constraint templates; healthcare and legal receive fewer datasets because constraint design in these domains is more sensitive and harder to standardize without introducing unrealistic or unsafe policy artifacts. Each dataset contains turn-level constraints and checkpoint prompts that force persistent style and policy retention across long dialogue horizons. Every conversation includes 10 checkpoints, producing aligned evaluation snapshots throughout the trajectory rather than only a final-turn score. At each checkpoint, systems are scored with a fixed LLM-as-judge template and then aggregated into both trajectory-level and end-of-run metrics. This protocol reduces overfitting to single-turn behavior and makes late-turn degradation directly visible.

5.2 Compared Systems

All four systems are executed under the same evaluation schedule so that differences can be attributed to context-management strategy rather than protocol variation. The baseline set intentionally spans the main design families used in practice: full-history prompting (strong recall, high cost), retrieval-only compression (lower context load, weaker guaranteed salience), and retrieval with summary compression (lower cost with drift risk).

- HierMem: constraint-first hierarchical curation
- Raw LLM: full-history baseline
- RAG: retrieval baseline
- RAG Summary: retrieval + rolling summary baseline

The RAG and RAG Summary baselines are intentionally lightweight single-stage systems. They retrieve full-turn records by embedding similarity and do not use turn chunking, reranking, or query rewriting. That makes them a clean lower-bound reference for architectural comparison, but they should not be interpreted as fully equipped retrieval stacks.

To preserve fairness, model-facing prompts and checkpoint structure are kept consistent, and analysis is reported at both aggregate and per-domain levels. This avoids misleading conclusions caused by domain imbalance or isolated strong cases.

5.3 Primary Metrics

The metric set is intentionally mixed: it combines output quality, constraint reliability, and runtime efficiency so that no single axis can dominate the conclusion.

- **Judge score**: weighted quality score at checkpoints.
- **Constraint survival**: fraction of checkpoints that satisfy active constraints.
- **Compute cost per turn**: token and retrieval operation cost proxy.
- **Session compute cost**: aggregate per-conversation compute cost.
- **Latency**: observed wall-clock response time per turn.

Judge score captures response usefulness and coherence under the active task state. Constraint survival captures whether the model actually obeys conversation-local rules over time. Cost and latency capture runtime pressure. Reporting all four together prevents a common evaluation failure mode where one method appears strong only because the benchmark ignores economic or adherence constraints.

5.4 Compute-Cost Estimation Protocol

Cost figures in this paper are estimated from runtime telemetry rather than billed provider invoices. This is necessary because benchmark runs include local execution and mixed-provider paths. For each turn, I combine token components and retrieval-operation components into relative cost units:

$$C_{\text{turn}} = C_{\text{token}} + C_{\text{retrieval}}, \tag{8}$$

where token cost is computed from measured/estimated main-model, curator/post-processing, and embedding-token counts, and retrieval cost is a lightweight proxy derived from retrieval operations (semantic queries and chunk fetches). Session compute cost is the sum across turns:

$$C_{\text{session}} = \sum_{t=1}^{T} C_{\text{turn},t}. \tag{9}$$

All values are dimensionless relative cost units under a shared token-weighting scheme. They are therefore directly comparable across systems and reproducible from the committed metric logs (`arch_metrics_research.json`), but should not be interpreted as dollar amounts without applying a provider-specific price table.

To estimate costs for other models/providers, the same token traces can be replayed with alternate price tables (per-million token rates and retrieval-unit rates). This allows hardware-agnostic what-if analysis without rerunning the full benchmark.

5.5 LLM-as-Judge Protocol Transparency

To make the evaluation pipeline auditable, I release the judge rubric prompt and scoring structure used during benchmark runs. Checkpoint scoring uses Gemini 3.1 Pro as the judge model and applies a fixed weighted rubric:

$$\text{JudgeScore} = 0.50 \cdot \text{CA} + 0.30 \cdot \text{RQ} + 0.20 \cdot \text{CC}, \tag{10}$$

where CA is constraint adherence, RQ is response quality/accuracy, and CC is conversational coherence/memory.

The published judge prompt also includes anti-vagueness and domain-drift penalties, plus a fairness rule that grants partial credit when a system attempts the requested logic but misses a domain-specific implementation detail. This is intended to reduce over-rewarding generic responses and to avoid collapsing nuanced attempts into zero-credit outcomes.

Judge prompt artifact: GitHub prompt file (v1.0.3-paper).

The benchmark dataset and updates are released through Hugging Face (hiermem-constraint-tracking).

6 Results

6.1 Overall Aggregate Across 15 Datasets

System	Judge ↑	Cost/turn ↓	Session cost ↓	Survival ↑
HierMem	**8.461**	**0.0069**	**0.334**	**0.933**
Raw LLM	6.908	0.0106	0.529	0.740
RAG	6.439	0.0102	0.466	0.667
RAG Summary	7.148	0.0105	0.483	0.760

Table 1: Aggregate metrics across all 15 datasets from the finalized Architecture C benchmark run. Cost/turn and Session cost are in relative dimensionless units (see Section **??**); values match `arch_metrics_research.json` in the released repository.

Key takeaways. Table 1 shows that HierMem leads all systems on the main tri-objective frontier: highest quality and survival with lowest cost. Compared with Raw LLM, quality improves by +1.553 points while session compute cost drops by 36.9%. Compared with retrieval baselines, HierMem maintains a stronger adherence profile while also reducing cost.

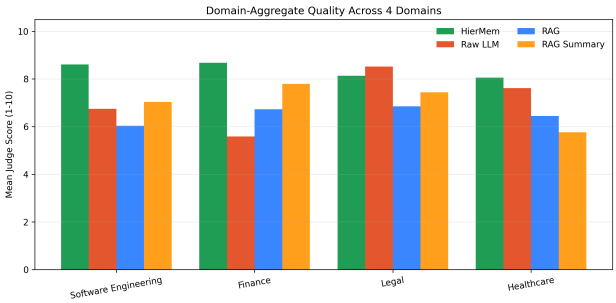
6.2 All-4-Domain Aggregate Evidence

The aggregate analysis across all four domains is the primary evidence in this paper because it avoids cherry-picking and captures how the system behaves under mixed workload characteristics. I evaluate quality, survival, and cost jointly: quality reflects output usefulness, survival reflects strict adherence to active constraints, and cost reflects deployment practicality.

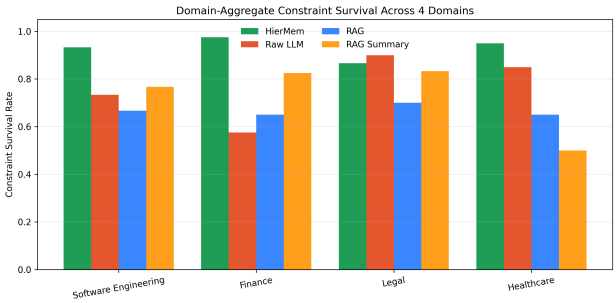
Domain aggregation in Fig. 2 shows HierMem leading quality, survival, and cost in software engineering, finance, and healthcare. In legal datasets, Raw LLM is slightly higher on quality and survival while HierMem remains lower-cost. This asymmetry is informative rather than problematic: legal interactions often include high-precision clause-level language where aggressive compaction can remove subtle but governing phrasing. The figure therefore supports a more nuanced conclusion than a single headline number: the architecture is broadly superior, but the legal domain warrants domain-specific caution and possibly less aggressive compression.

6.3 Global Dynamics and Diagnostics

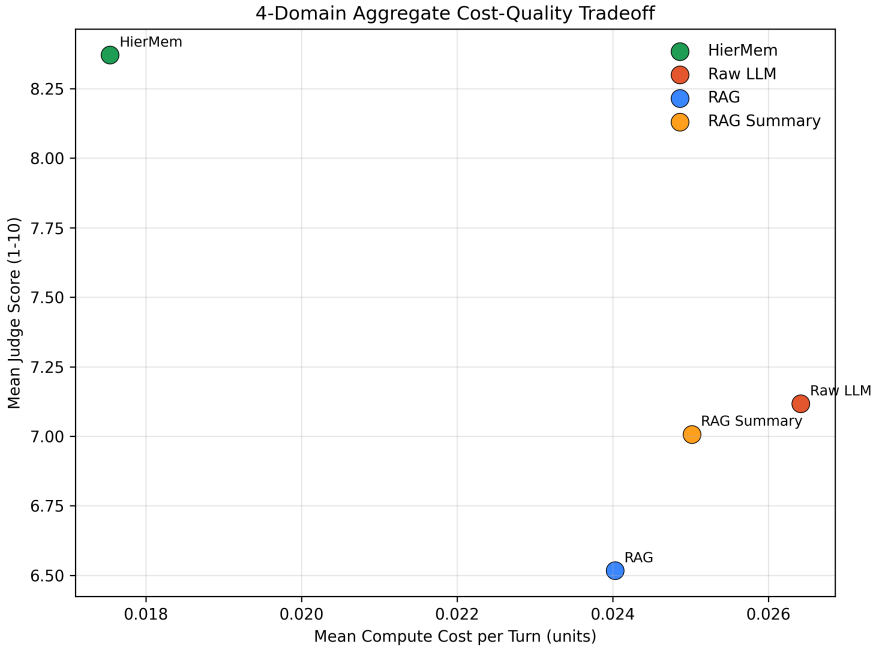
Beyond end-point aggregates, I analyze global trajectory behavior over checkpoints and turns. This is important because some methods look competitive at early checkpoints but degrade later as



(a) Domain aggregate quality



(b) Domain aggregate survival



(c) Domain aggregate Pareto tradeoff

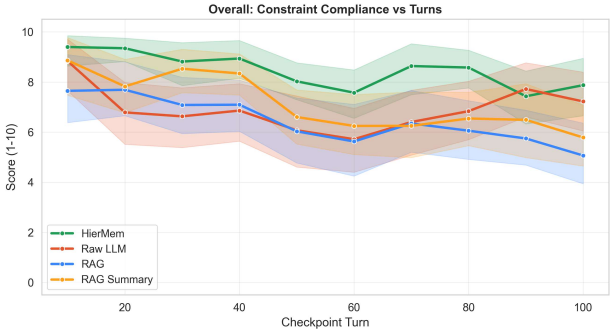
Figure 2: All-4-domain aggregate evidence used as the primary evaluation view.

context pressure increases. I therefore inspect quality trend stability, global Pareto position, explicit cost composition, and context payload together.

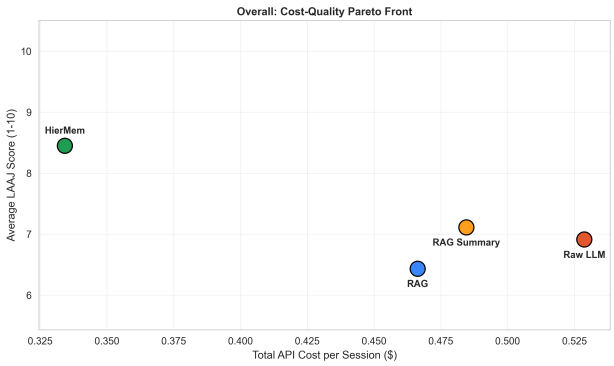
The global trends in Fig. 3 show that HierMem avoids severe late-turn degradation while keeping context payload growth bounded. The context-complexity plot is more informative than raw latency for the architecture argument because it directly measures prompt size, which drives both attention dilution and compute burden. Cost breakdowns indicate that curator/post-processing overhead is present but dominated by savings from reduced expensive main-model context load.

Crucially, Figure 4 visualizes the system's resilience to *context pressure*. By turn 50, a typical benchmark conversation accumulates nearly 80,000 tokens of raw history. While the Raw LLM hits a hard 32k KV-cache ceiling by turn 25 (leading to forced truncation and constraint collapse), HierMem maintains a stable prompt payload of roughly 16k to 23k tokens. This represents a measured compression ratio of 4.7x without loss of functional reasoning, proving the system is not merely accurate at the end of a run, but actively insulates the main LLM from sequential capacity overloads.

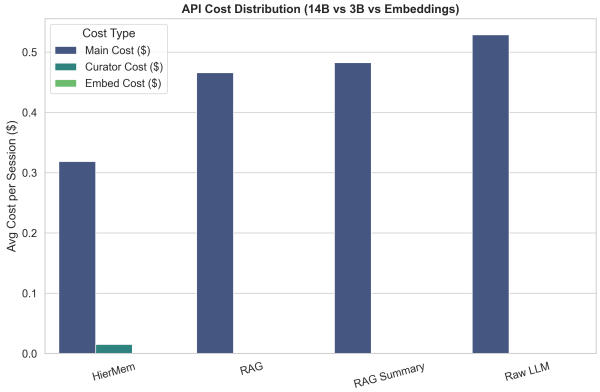
This section is also where the paper makes its strongest system-level argument: the architecture



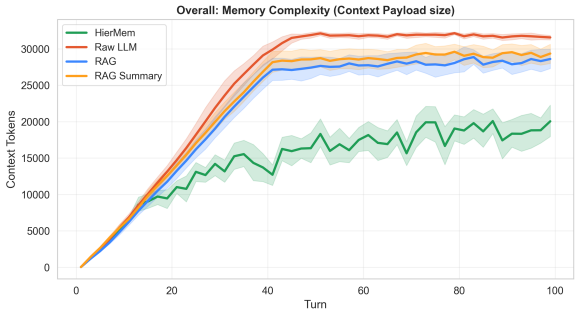
(a) Overall quality trend



(b) Overall Pareto view

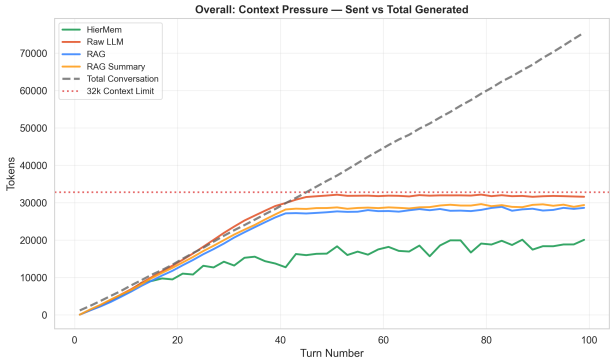


(c) Overall cost breakdown

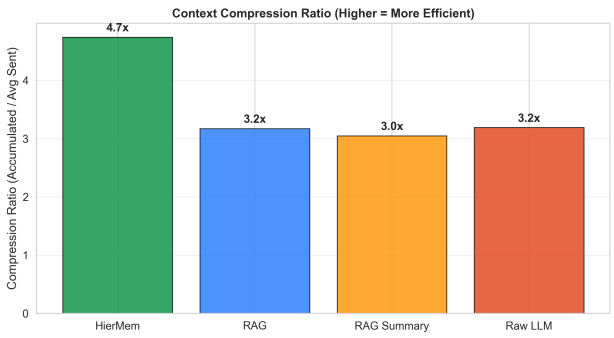


(d) Overall memory complexity / context payload

Figure 3: Global dynamics across quality, cost, and context payload.



(a) Context pressure relative to accumulated text



(b) Average architectural compression ratio

Figure 4: Context Pressure: Demonstrating how HierMem manages massive conversational accumulation compared to raw context window ceilings. The dashed line represents total accumulated conversational history (user queries and assistant responses).

is not just improving final-turn quality, but preserving a usable operating envelope across the full session. That distinction matters because a system that is strong only at the end of a benchmark is much less useful in deployment than one whose quality-cost profile remains stable turn after turn.

6.4 Implementation Evidence: Curator Output and Routing

To keep the paper grounded in actual runtime behavior, I include a representative curator decision from the finalized benchmark logs. In one software-engineering conversation, the curator selected a hybrid strategy because the user asked a Kafka-to-Snowflake question that was not directly covered by the existing segment labels. The runtime then combined segment fetches, semantic queries, and recent-turn context:


```
{
    "curator_strategy": "HYBRID",
    "segments_fetched": ["seg_001", "seg_003"],
    "peripheral_segments": [
        "seg_001: Turn 5-7: The user inquires about Lambda/Kappa differences.",
        "seg_003: Turn 4-6: The assistant provides a brief overview of dbt."
    ],
    "semantic_queries": [
        "how close can we get to exactly-once from kafka to snowflake",
        "kafka to snowflake exactly-once implementation",
        "stream processing exactly-once guarantees using kafka and snowflake"
    ],
    "curator_reasoning": "The prompt is related to Kafka and Snowflake, which are not directly
}
```


This trace is useful for two reasons. First, it shows that the curator is not a blind semantic retriever: it can mix L1 segment fetches, L2/L3 semantic search, and peripheral awareness in one decision. Second, it shows that routing is relational, not only lexical. The curator included old segments because they were structurally helpful, not because they were exact semantic matches.

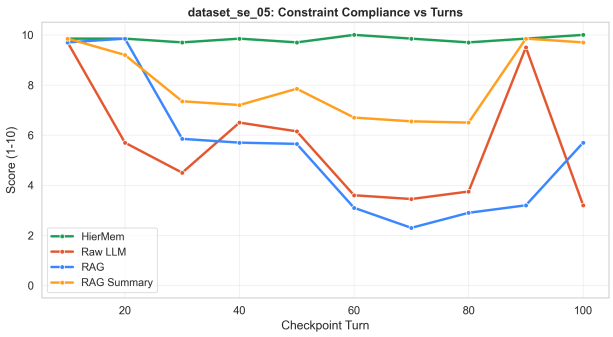
The full benchmark logs also expose pipeline details such as prompt-token estimates, recent-turn inclusion, and zone token breakdown. That makes the runtime auditable at the same granularity as the reported aggregate metrics.

6.5 Prompt and Judge Protocol Sketch

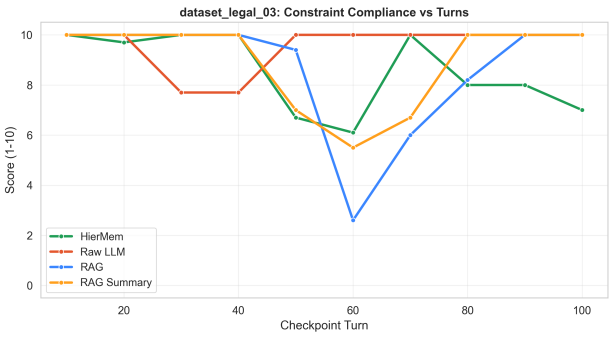
The actual prompt structure matters, so I summarize the benchmark prompt layout rather than treating it as a hidden implementation detail. The curator prompt receives three blocks only: the current user message, the L0 topic directory, and the active constraints. It must return JSON with retrieval strategy, segments to fetch, semantic queries, peripheral segments, full-turn requests, and a short reasoning field. The main assembler then receives a fixed zone order: constraints, recent turns, retrieved context, peripheral summaries, and current request.

This five-block layout is the canonical one used in the figure and the runtime code. The paper uses the shorter label “zone” for the major salience regions, but recent turns are always a distinct high-priority block in the actual assembler.

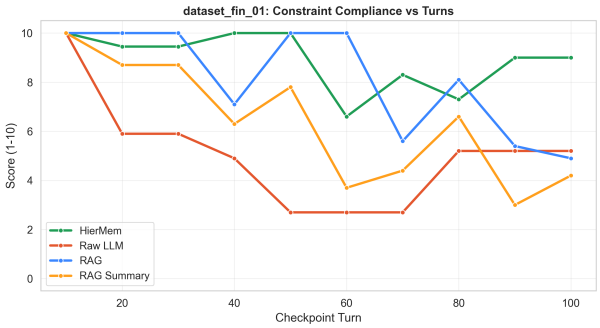
The judge prompt follows the same philosophy. It scores checkpoint outputs against a fixed rubric and asks for a single structured decision rather than free-form commentary. In the paper, I intentionally keep the rubric simple and fixed so that comparisons are about memory behavior, not about prompt artistry.



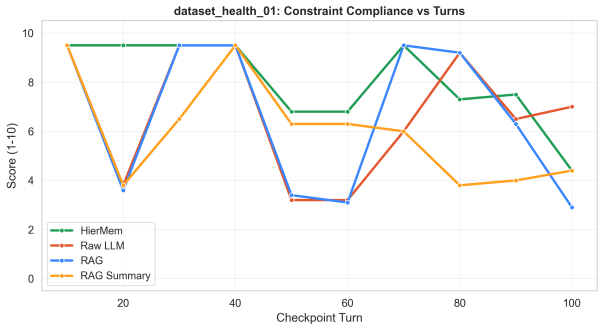
(a) Software engineering diagnostic



(b) Legal diagnostic



(c) Finance diagnostic



(d) Healthcare diagnostic

Figure 5: Single-dataset diagnostics with all four domains represented (software engineering: dataset_se_05, legal: dataset_legal_03, finance: dataset_fin_01, healthcare: dataset_health_01). These support mechanism analysis, while all-4-domain aggregates remain primary evidence.

6.6 Single-Dataset Diagnostic View

Single-dataset diagnostics are treated as mechanism analysis, not primary claims. To ensure domain coverage, this subsection includes one diagnostic item per domain: software engineering and legal are shown with explicit per-dataset plots, while finance and healthcare are now represented by real benchmark plots copied into the paper figures folder. These cases let me point to concrete domain-specific behavior rather than relying only on aggregate averages.

Figure 5 provides concrete per-domain trajectories and helps connect the aggregate story to dataset-level behavior, especially in domains where formatting strictness and legal precision create different error profiles. Finance mainly reinforces the cost-and-survival trend seen in the aggregates, while healthcare shows that the architecture remains stable even in a domain with more safety- and terminology-sensitive responses.

7 Discussion

Mechanism-level interpretation. The gains are not explained by retrieval volume alone. The critical difference is *placement policy*: constraints are always injected in a protected zone, while retrieved content remains selective and budgeted. This avoids the frequent failure mode where retrieval returns relevant but non-governing text and the model drifts from prior obligations. In other words, HierMem improves not only memory access but memory *governance*: the model consistently sees what it must obey before it sees what is merely useful.

Why this is not just "RAG with extra steps." RAG decides *what to fetch* mostly from relevance signals (for example embedding similarity), while HierMem jointly decides what to fetch, what to keep peripheral, and what must be foregrounded as non-evictable policy context. In long sessions, this placement and routing control is often the difference between recalling a rule and following it. The curator can include context that is relationally necessary (for example dependency turns and rule-setting turns), not only semantically nearest chunks. This is especially visible in adherence metrics, where retrieval-only methods can recover relevant text but still underperform if the assembled prompt does not preserve constraint salience.

Cost-quality dynamics. A smaller curator introduces additional calls but reduces the largest cost driver: high-token main-model prompts. The benchmark results indicate that this trade is favorable in aggregate. The key systems insight is that not all calls have equal economic weight: several lightweight orchestration calls can still be cheaper than repeatedly sending large contexts to a premium generator. The pipeline therefore provides a practical path to improving both reliability and efficiency rather than forcing an either-or compromise.

Legal-domain edge case. The legal-domain result highlights an important nuance: for some high-precision textual tasks, heavy compaction may be suboptimal. Future work should include domain-conditioned compression policies, legal-specific retrieval anchors, and ablations that tune passthrough thresholds by domain. A promising direction is confidence-triggered fallback, where the system dynamically increases raw-context passthrough when legal obligation patterns are detected.

8 Limitations and Threats to Validity

This study has several limitations that should guide interpretation. First, all reported results correspond to one finalized runtime configuration (Architecture C with a Qwen2.5-14B main model). While this isolates the effect of the proposed design, it does not fully characterize sensitivity across model families and context-window scales. Second, parts of cost accounting rely on modeled components for operations where provider-side billing metadata is not uniformly exposed; this can introduce small estimation error. Third, LLM-as-judge evaluation, although practical and now widely used, remains sensitive to prompt template and judge-model identity. Fourth, domain coverage is intentionally broad but imbalanced in sample count, with heavier software representation than healthcare. Fifth, the benchmark corpus is synthetic and controlled; therefore, conclusions in this paper are benchmark-scoped and should not be interpreted as direct estimates of production user behavior without additional real-world validation. Finally, I do not include full ablation tables in this manuscript version, so readers cannot yet isolate the exact marginal contribution of each runtime component under a single unified table.

9 Reproducibility

The repository includes benchmark execution, metric aggregation, and plotting scripts, enabling end-to-end regeneration of the presented figures and tables from run directories. A typical workflow is to run the benchmark pipeline first, compute research metrics over produced logs, and then render paper-ready plots from the same root. This sequence is intentionally linear to reduce accidental mismatch between metric files and figure artifacts.

Artifact links. To support reproducibility and citation stability, I provide both a frozen code snapshot and a dataset release endpoint:

- Code release tag (paper snapshot): GitHub release v1.0.3-paper
- Dataset release page (Hugging Face): HF dataset card
- Dataset files/version browser: HF files and versions

The code tag should correspond to the exact commit used to generate paper figures and tables.

The dataset artifact itself contains the canonical release bundle used in this paper, including: conversation JSON files, checkpoint test structures, dataset manifest/checksums, and publication metadata. This is intended to keep evaluation provenance visible directly at the dataset endpoint without requiring readers to infer file structure from repository internals.

For stronger camera-ready reproducibility, three additions are recommended: (i) immutable environment capture (package lock plus model endpoint metadata), (ii) judge prompt and version hashing, and (iii) explicit per-domain ablation manifests. Together, these additions would make cross-lab replication and longitudinal comparison substantially easier.

The evaluation bundle should also preserve the judge model configuration explicitly, including the Gemini 3.1 Pro settings used for scoring, so the exact benchmark protocol remains replayable.

10 Conclusion

HierMem treats long-horizon assistant reliability as a systems problem rather than a pure model-capacity problem. By combining hierarchical memory, adaptive mode switching, and invariant constraint placement, it achieves strong quality, adherence, and cost performance simultaneously across 15 long conversations. The all-4-domain aggregate analysis supports robust gains in software engineering, finance, and healthcare while exposing a meaningful legal-domain edge case that clarifies where domain-aware policy adaptation is needed.

The broader implication is that reliable long-horizon behavior can be engineered through explicit runtime structure, not only larger context windows or larger models. In practice, this means better controllability for real assistants: constraints remain visible, context remains bounded, and cost remains predictable. The next research step is to make this architecture domain-adaptive with transparent ablations, so that the same reliability principles generalize across specialized high-precision workloads and can be validated on real-world conversation logs in addition to the current synthetic benchmark setting.

Author Contributions: Y.D. conceptualized the architecture, conducted the experiments, analyzed the data, and wrote the manuscript.
Funding: This research received no external funding.
Data Availability Statement: The benchmark dataset, evaluation results, and codebase are publicly available at https://huggingface.co/datasets/yashdoke7/hiermem-constraint-tracking and https://github.com/yashdoke7/llm-hiermem.
Conflicts of Interest: The author declares no conflicts of interest.

References

[1] Patrick Lewis et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*, 2020.

[2] Gautier Izacard et al. Atlas: Few-Shot Learning with Retrieval Augmented Language Models. *JMLR*, 23(251):1–43, 2022.

[3] Sebastian Borgeaud et al. Improving Language Models by Retrieving from Trillions of Tokens. In *ICML*, 2022.

[4] Nelson F. Liu et al. Lost in the Middle: How Language Models Use Long Contexts. *TACL*, 2023.

[5] Charles Packer et al. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*, 2023.

[6] Akari Asai et al. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. *arXiv preprint arXiv:2310.11511*, 2023.

[7] OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

[8] Anthropic. The Claude 3 Model Family: Opus, Sonnet, Haiku. Technical report, 2024.

[9] Google DeepMind. Gemini 1.5: Unlocking Multimodal Understanding across Millions of Tokens of Context. Technical report, 2024.

[10] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The Long-Document Transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[11] Manzil Zaheer et al. Big Bird: Transformers for Longer Sequences. In *NeurIPS*, 2020.

[12] Ofir Press, Noah A. Smith, and Mike Lewis. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. In *ICLR*, 2022.

[13] Lianmin Zheng et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *NeurIPS*, 2023.