MEMONDEMAND: A Memory Management System for Large-Scale Enterprise Data

Xinyuan Song[1,2] Bowen Zhu[1] Hasibul Haque[1] Liang Zhao[1,2]
[1]Causal Dynamics Lab, USA [2]Emory University, USA

Abstract

Enterprise repositories are large, heterogeneous, and continuously updated, making retrieval difficult when efficient access, source-faithful evidence, and cross-query adaptation must be supported together. Enterprise memory extends retrieval beyond the model context, but existing systems do not jointly address collection-specific hierarchy construction, low-cost routing, detailed evidence loading, and workload-aware memory updates at this scale. We introduce MEMONDEMAND, short for *On-Demand Memory*, a memory management system with three coordinated mechanisms: a dynamic multi-level hierarchy that determines the abstraction structure and depth for each collection, dual memory at every hierarchy level that separates distilled routing from detailed evidence, and on-demand memory promotion that updates node priority under a bounded active-state budget. On EnterpriseRAG-Bench, MEMONDEMAND outperforms the strongest published LB#1 result at every evaluated scale from 10M tokens through the complete 618M-token collection, with gains of 12.23% at 10M and 4.66% at 618M. Results on FinanceBench, HotpotQA, and FRAMES further show strong performance across financial, multi-hop, and fact-retrieval settings. Together, these results establish MEMONDEMAND as an accurate, efficient, and scalable memory solution for very large enterprise repositories across data scales, domains, and evidence requirements. Our code is available at `https://github.com/xfab-xinyuansong/MemOnDemand.git`.

1 Introduction

Enterprise repositories contain policies, contracts, email, tables, tickets, code, and records produced by many teams and software systems. In large organizations, these repositories can reach hundreds of millions or billions of searchable items and continue to grow as new records, revisions, and derived data are added (Chang et al., 2006; O'Neil et al., 1996; Jayaram Subramanya et al., 2019; Wang et al., 2021; Armbrust et al., 2021). Their contents differ in format, scope, and authority: a policy may define a general rule, a contract may specify an exception, an email may record a later decision, and a ticket or table may contain the identifier needed to apply that decision (Armbrust et al., 2021; Codd, 1970). Their structure also changes with departments, data sources, access patterns, and document versions (O'Neil et al., 1996; Chaudhuri and Narasayya, 2007). Enterprise retrieval must therefore search a large, heterogeneous, and changing collection while preserving exact records and traceable source identities. Approximate similarity indices make large candidate spaces searchable (Malkov and Yashunin, 2020; Jayaram Subramanya et al., 2019; Wang et al., 2021), but they do not define how records should be organized, represented, loaded for answering, or maintained across queries.

Enterprise memory makes information from large repositories available beyond the model context and reusable across queries (Lewis et al., 2020; Wang et al., 2023). Existing systems retrieve interaction histories (Zhong et al., 2023), derive higher-level reflections (Park et al., 2023), manage active and external memory tiers (Packer et al., 2023), and support memory extraction, organization, updating, and forgetting (Chhikara et al., 2025; Xu et al., 2025; Kang et al., 2025; Li et al., 2025). However, enterprise memory must scale to hundreds of millions or billions of items while preserving source identity, update consistency, and auditable evidence use. Compressed memories may omit answer-critical details (Jiang et al., 2023, 2024; Xu et al., 2024; Packer et al., 2023), stored memories may become stale (Wang et al., 2023), and most agent-memory systems do not target source-level citation over large multi-source repositories (Park et al., 2023; Zhong et al., 2023; Chhikara et al., 2025; Xu et al., 2025). Long-context models also

remain sensitive to irrelevant context (Liu et al., 2024; Hsieh et al., 2024). Enterprise memory must therefore support large-scale retrieval and reuse without allowing compressed or stale memories to replace authoritative sources.

The heterogeneous structure of enterprise data motivates hierarchical memory, where high-level records support routing and lower-level records retain source-specific evidence. Prior work studies recursive, graph-based, and multi-level retrieval (Sarthi et al., 2024; Edge et al., 2024; Gutiérrez et al., 2024; Qian et al., 2024; Sun and Zeng, 2025). However, enterprise collections differ across domains and tenants, so a fixed hierarchy or predefined taxonomy cannot fit all settings. Enterprise memory should instead infer both the hierarchy and its depth from each collection, while preserving direct source-leaf access when higher-level abstractions omit relevant details (Jayaram Subramanya et al., 2019; Indyk and Xu, 2023).

Repository scale also requires separating memory for routing from memory for evidence. Detailed memory preserves source-specific content but is expensive to search and load, while compressed or compact memory reduces cost but may omit answer-critical details (Jiang et al., 2023, 2024; Xu et al., 2024; Wang et al., 2023; Packer et al., 2023; Qian et al., 2024). Enterprise retrieval therefore needs dual representations: distilled memory for efficient routing and detailed memory for generation and citation. Both must resolve to the same source ID so that compressed records guide access without replacing the authoritative source (Codd, 1970; Abadi et al., 2007).

Enterprise memory must also adapt to changing data and workloads. Eagerly preparing all representations wastes computation and storage on sources that may never be used, while adaptive data systems update access structures according to observed demand (Idreos et al., 2007; Megiddo and Modha, 2003; Chaudhuri and Narasayya, 2007; O'Neil et al., 1996). Enterprise memory should therefore promote useful sources on demand, refresh repeatedly accessed records, and demote stale or low-value ones under a fixed capacity, without bypassing source validation or evidence-loading constraints.

We introduce MEMONDEMAND to address these three problems through a unified memory management design. First, *dynamic multi-level hierarchy* determines both the abstraction structure and the number of levels for each collection, allowing the same system to adapt across domains and tenants while retaining direct access to L0 source nodes. Second, *dual memory at each hierarchy level* separates retrieval from answering: distilled memory supports efficient routing, while selected detailed memory provides the evidence used for generation and citation. Third, *on-demand memory promotion* adjusts node priority from observed use, refreshes repeatedly accessed nodes, and demotes stale or low-value nodes under a bounded active-state budget. Together, these mechanisms improve retrieval, ranking, and evidence selection by coordinating hierarchy navigation, compact routing, detailed evidence loading, and cross-query adaptation within one memory manager.

Figures 1–3 illustrate the three mechanisms and their interaction. We evaluate MEMONDEMAND on EnterpriseRAG-Bench, a large-scale enterprise retrieval benchmark with approximately 500,000 documents (Sun et al., 2026). Against LB#1, the strongest published solution on this benchmark, MEMONDEMAND improves Combined by 12.23% at 10M source tokens and remains 4.66% higher on the complete 618M-token collection. Dual-memory selective loading reduces answer-input tokens by 70.2% relative to detailed-only loading, while the persisted hierarchy becomes query-ready in 1.46 seconds rather than the estimated 593.5 seconds required for eager preparation. Results on FinanceBench, HotpotQA, and FRAMES further show strong performance across financial, multi-hop, and fact-retrieval settings. Together, these results establish MEMONDEMAND as an accurate, efficient, and scalable memory solution for very large enterprise repositories.

The main contributions of this work are summarized as follows:

- We introduce MEMONDEMAND, a memory management system for large-scale enterprise retrieval that jointly manages hierarchical organization, retrieval representation, evidence loading, and cross-query memory state.
- We develop three coordinated mechanisms: a dynamic multi-level hierarchy that adapts its depth and abstractions to each collection, dual memory at every hierarchy level that separates efficient routing from source-faithful answering, and on-demand memory promotion that adapts node priority to changing workloads under a bounded active-state budget.
- We demonstrate that a managed memory sys-

tem can operate successfully at ultra-large scale: on the complete 618M-token EnterpriseRAG memory, MEMONDEMAND reaches 72.88% Combined, 4.66% above the published LB#1 reference. Across scales from 10M tokens to 618M and on three external benchmarks, it also reduces answer-input cost, improves query readiness, and maintains strong performance on FinanceBench, HotpotQA, and FRAMES.

2 Problem Formulation

We consider retrieval over a large enterprise collection $\mathcal{D} = \{d_i\}_{i=1}^{n}$ for a sequence of queries $\mathcal{Q} = (q_1, q_2, \ldots)$. Each source preserves its original content and identity, while the memory system may construct additional representations and states to support retrieval (Codd, 1970; Abadi et al., 2007).

Given a query q_t, the system retrieves an evidence set $E_t \subseteq \mathcal{D}$ and generates an answer a_t with citations C_t. The answer must satisfy

$$C_t \subseteq \mathrm{ID}(E_t), \qquad T(q_t, E_t) \leq B,$$

where B is the answer-input budget. Thus, every citation must refer to evidence provided to the answer model, and the selected evidence must remain within the available context.

The main problem is to make retrieval accurate and efficient as the collection grows and changes. This requires a hierarchy that can adapt to different collections, compact memory for efficient retrieval, detailed memory for answering, and cross-query updates that prioritize useful information without retaining unlimited state. These requirements are important because increasing the amount of retrieved context does not guarantee that the model will identify and use the decisive evidence (Liu et al., 2024; Hsieh et al., 2024).

3 MEMONDEMAND

3.1 System Contract and Execution

MEMONDEMAND maintains three forms of state aligned with its three mechanisms: a dynamic hierarchy over memory nodes, distilled and detailed memory at each node, and cross-query promotion state. Each node records its hierarchy level, two memory representations, token cost, child relations, and current promotion status. This design follows the database principle that multiple access paths and derived structures should remain tied to the same underlying record (Codd, 1970; Abadi et al., 2007).

Given a query, MEMONDEMAND performs coarse-to-fine retrieval over distilled memory, starting from the highest available level and descending from L_ℓ to $L_{\ell-1}$, $L_{\ell-2}$, and finally L0 (Sarthi et al., 2024; Edge et al., 2024). Sentence embeddings support efficient search at each level (Reimers and Gurevych, 2019). The retrieved L0 candidates are then ranked with promotion signals, after which detailed memory is loaded under the answer-input budget B. Only the selected detailed memory is provided to the answer model, keeping generation independent from the retrieval and storage components (Lewis et al., 2020).

This execution order assigns a distinct role to each mechanism: the dynamic hierarchy organizes and narrows retrieval, dual memory separates routing from evidence, and on-demand promotion adapts candidate priority across queries. Algorithm 1 summarizes the complete pipeline, and the following subsections describe each mechanism.

3.2 Dynamic Multi-Level Hierarchy

MEMONDEMAND organizes enterprise memory as a dynamic sequence of abstraction levels. Level L0 contains the original source-level records and their distilled routing representations. Level L1 groups related L0 records and forms the first abstraction layer, while L2 summarizes related L1 records at a higher semantic level. The same process can continue to L3, L4, and beyond when the collection supports further abstraction. Rather than fixing the hierarchy depth in advance, MEMONDEMAND decides both how to group records at each level and whether another higher level should be created.

Starting from level L_ℓ, MEMONDEMAND clusters the distilled representations of its records and forms a proposal for level $L_{\ell+1}$. The proposal includes group-size statistics, cluster coherence, residual noise, sampled content, and the remaining construction budget. An LLM controller then returns CREATE, RECLUSTER, or STOP. CREATE accepts the proposed grouping and constructs $L_{\ell+1}$, RECLUSTER revises the grouping before reevaluation, and STOP ends construction when a higher abstraction level is not supported by the current data. This procedure determines the hierarchy structure and depth separately for each collection, allowing different domains and tenants to produce different numbers of levels and different abstractions (Sarthi

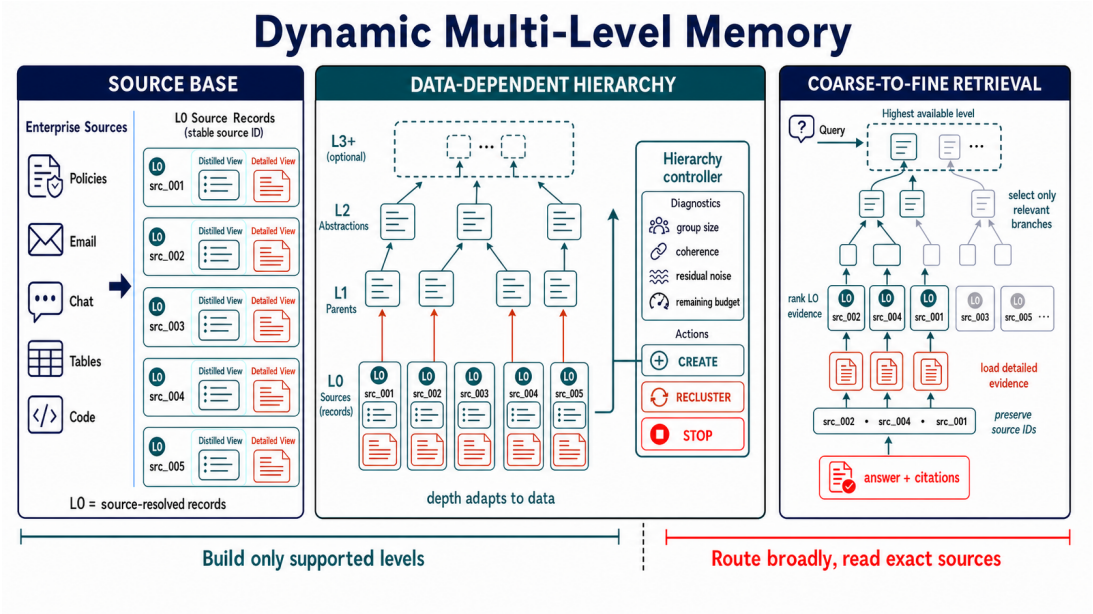

Figure 1: **Dynamic multi-level hierarchy construction.** L0 stores source-level records, L1 and L2 provide increasingly higher abstractions, and the controller decides whether to create, revise, or stop each additional level.

et al., 2024; Edge et al., 2024). Each accepted parent record retains links to all descendant source records. Figure 1 illustrates the construction process, Table 9 reports the resulting hierarchy sizes, and Algorithm 2 gives the full procedure. GPT-5.4 mini produces the L0 distilled records, while GPT-5.4 evaluates and constructs higher abstraction levels.

At query time, MEMONDEMAND begins at the highest available level L_ℓ and retrieves the records most relevant to the query. The controller then evaluates whether the selected records provide sufficient information to continue routing. If not, MEMONDEMAND follows their child links to level $L_{\ell-1}$, performs retrieval again within those selected branches, and repeats the same decision. The process proceeds from L_ℓ to $L_{\ell-1}$, $L_{\ell-2}$, and so on until it reaches L0. At L0, the retrieved source records provide the detailed evidence used for final ranking, generation, and citation.

3.3 Dual Memory at Each Hierarchy Level

Every hierarchy node, from L0 source nodes to higher-level abstraction nodes, maintains two complementary representations: *distilled memory* and *detailed memory*. Distilled memory is a compact retrieval representation that preserves the main topic, entities, identifiers, and decision-relevant facts of the node. Detailed memory retains the full information represented by that node: the original source content at L0 and the complete abstraction constructed from child nodes at higher levels. This design allows MEMONDEMAND to search compact memory throughout the hierarchy and load richer content only when it is needed for answering (Jiang et al., 2023, 2024; Xu et al., 2024).

For a node v at level L_ℓ, MEMONDEMAND stores

$$v = (\ell_v, b_v, d_v, \tau_v^b, \tau_v^d), \tag{1}$$

where ℓ_v is its hierarchy level, b_v and d_v are its distilled and detailed memories, and τ_v^b and τ_v^d are their token counts. The two representations are indexed and accessed separately. During hierarchical retrieval, MEMONDEMAND searches distilled memory at level L_ℓ, selects relevant nodes, and descends to their children at $L_{\ell-1}$. Detailed memory is not loaded during this routing process, avoiding the token cost of repeatedly reading full node content across hierarchy levels.

After retrieval reaches L0, MEMONDEMAND combines signals from all hierarchy paths, ranks the resulting source candidates, and loads detailed memory only for the highest-ranked sources that fit the answer-input budget. Distilled memory guides retrieval, while detailed memory provides the evidence used for generation and citation. This separation avoids repeatedly loading full content during hierarchy traversal and therefore reduces token consumption (Abadi et al., 2007). When multiple routes reach the same L0 source, their signals are fused into a single candidate score before detailed

memory is loaded. Figure 2 illustrates how distilled memory supports hierarchical retrieval while only selected detailed memories enter generation and citation.

Algorithm 3 summarizes the execution order: retrieve distilled memory across hierarchy levels, merge signals for each L0 source, rank the resulting candidates, and load detailed memory until the answer-input budget is reached.

3.4 On-Demand Memory Promotion

Enterprise workloads change over time, so the value of a memory node cannot be fixed at construction time (Chaudhuri and Narasayya, 2007; O'Neil et al., 1996). Preparing or prioritizing every node in advance wastes computation and storage on records that may never be used (Abadi et al., 2007; Idreos et al., 2007). MEMONDEMAND therefore updates memory priority only when query-time evidence indicates that a node is useful, avoiding unnecessary up-front preparation while adapting retrieval to observed demand (Idreos et al., 2007; Megiddo and Modha, 2003; Chaudhuri and Narasayya, 2007).

For each candidate node v at query step t, the promotion controller uses the query, hierarchy level, retrieval scores, node metadata, and recent usage state to produce a promotion score $g_t(v) \in [0, 1]$. If $g_t(v) \geq \theta$, the node is promoted, increasing its current retrieval score and recording its value for later queries. Repeated access refreshes the node, whereas inactive nodes gradually lose priority, allowing promotion to support both current-query selection and reusable cross-query state within the bounded retrieval process (Megiddo and Modha, 2003; Zhong et al., 2023). Promotion updates retrieval priority and cross-query state, but detailed memory is still loaded separately under the answer-input budget (Abadi et al., 2007; Packer et al., 2023).

Because only a limited number of nodes can remain active, MEMONDEMAND maintains a promotion budget K and enforces $|\,\mathrm{active}(\mathbf{z}_t)| \leq K$. Each promoted node receives a retention score

$$\rho_t(v) = \max\left(0, s_t(v) - \frac{t - t_{\mathrm{last}}(v)}{\tau}\right), \quad (2)$$

where $s_t(v)$ is the stored promotion score, $t_{\mathrm{last}}(v)$ is the most recent access time, and τ controls decay. Repeated use refreshes the score, while stale nodes gradually lose retention value. When the number of active nodes exceeds K, MEMONDEMAND demotes expired nodes first and then removes the lowest-retention nodes until the budget is restored. A demoted node can be promoted again when later queries make it useful. Each promotion, refresh, and demotion is written to an append-only log, making the evolution of memory priority traceable.

Figure 3 illustrates this lifecycle. Promotion reacts to observed query demand, avoids unnecessary advance preparation, and adapts candidate priority within a bounded online process. The active-node budget limits retained state, while decay removes stale or low-value nodes when capacity is reached. This process allows MEMONDEMAND to adapt retrieval across changing workloads without rebuilding the hierarchy or treating promoted nodes as answer evidence.

Algorithm 1 MEMONDEMAND construction and query execution

Require: sources $\mathcal{D}$, queries $\mathcal{Q}$, answer-input budget B
1: build L0 memory and dynamically construct higher abstraction levels
2: **for** $q_t \in \mathcal{Q}$ **do**
3: retrieve distilled memory from the highest level to L0
4: score candidate nodes and apply on-demand promotion
5: refresh reused nodes and demote nodes whose retention scores expire
6: rank L0 candidates and load detailed memory until budget B is reached
7: generate the answer from loaded detailed memory and log memory-state changes
8: **end for**
9: **return** answers, retrieved evidence, updated memory state, and audit events

System properties. Appendix E describes the retrieval, context-accounting, and memory-update properties of MEMONDEMAND. These properties specify how the three mechanisms interact during execution rather than introducing a separate theoretical objective.

4 Experimental Setup

Our primary evaluation uses EnterpriseRAG-Bench (ENTERPRISERAG), a synthetic benchmark for enterprise retrieval over heterogeneous sources, multi-source evidence, and citation-based answering (Sun et al., 2026). We evaluate seven collection sizes containing 10M, 20M, 60M, 100M, 150M, 250M, and the complete 618M source tokens. Each system answers the same 500 questions at every scale and returns both an answer and the source IDs used to support it. We additionally evaluate on FinanceBench for financial-document question

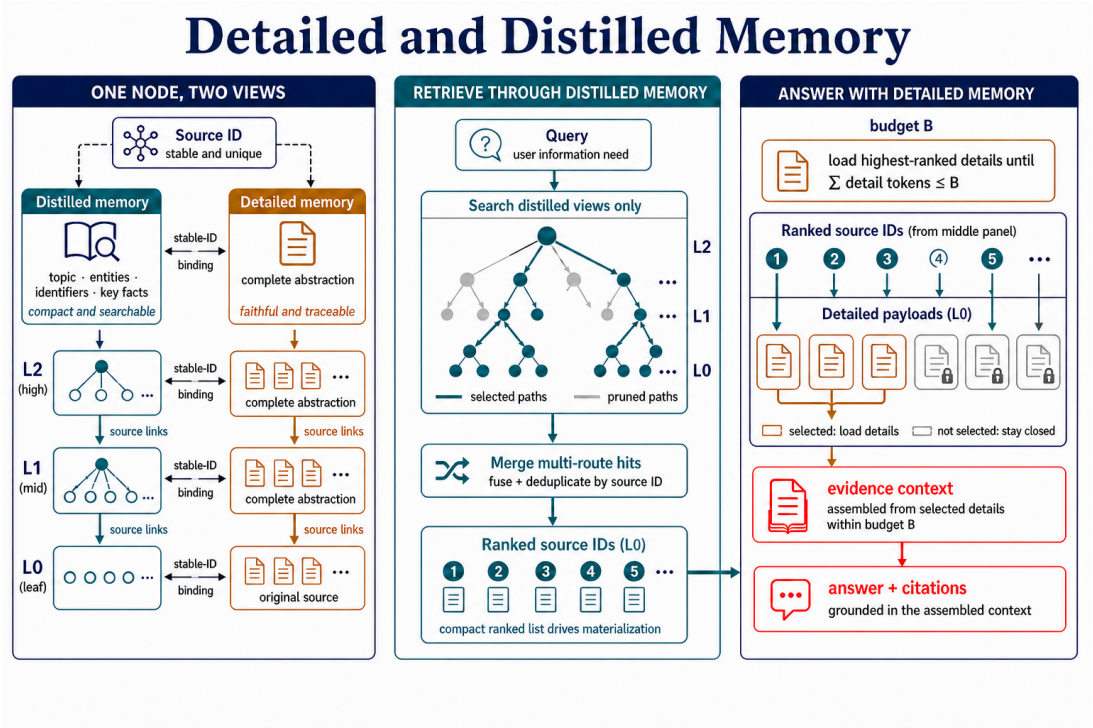

Figure 2: **Dual memory across hierarchy levels.** Each node stores distilled memory for low-cost retrieval and detailed memory for information preservation, while only selected detailed memories enter the answer context.

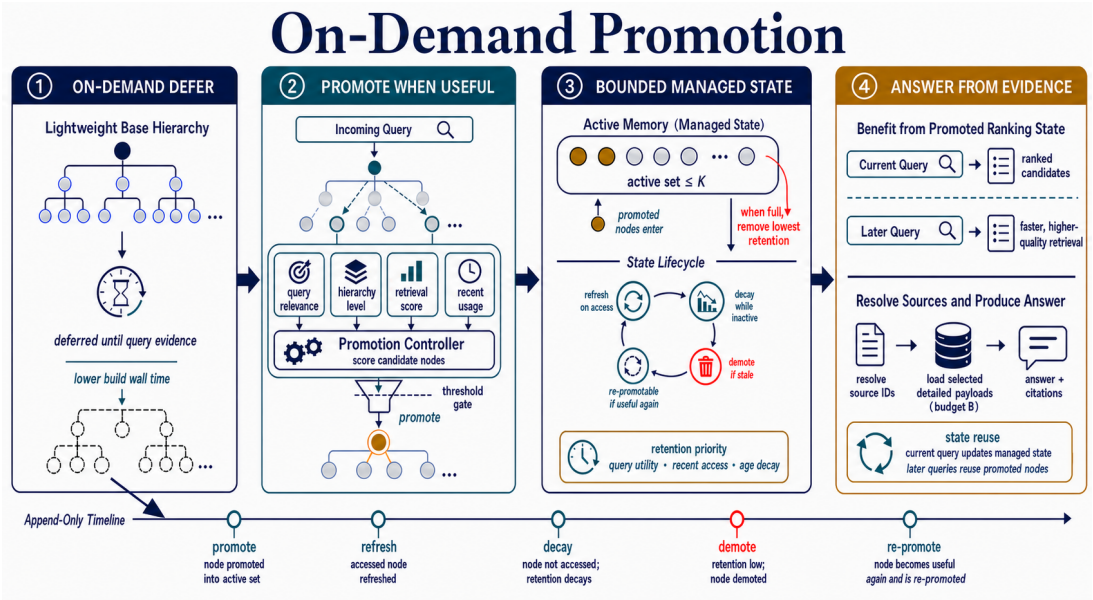

Figure 3: **On-demand memory promotion.** Query use promotes relevant nodes, repeated access refreshes them, and decay demotes stale nodes, while detailed memory is loaded separately for answering.

answering (Islam et al., 2023), HotpotQA for multi-hop question answering (Yang et al., 2018), and FRAMES for fact retrieval and reasoning over multiple sources (Krishna et al., 2025).

EnterpriseRAG-Bench metrics. Combined is the primary benchmark score and summarizes answer and evidence quality. Combined, Δ Combined, Correct, Complete, and Document Recall are reported as percentages, with Δ Combined denoting the absolute percentage difference from LB#1. Evidence F1 and Invalid Document Ratio (InvDoc) are reported on the $[0, 1]$ scale, where higher Evidence F1 and lower InvDoc are better. Promo and Demote report the total numbers of accepted promotion and demotion transitions

across the complete query run. Table 1 also includes LB#1, the strongest published ENTERPRISERAG result under the same 500-question protocol and a collection of approximately 500,000 documents (Sun et al., 2026). Additional implementation details, including model assignment, retrieval settings, evaluation rules, failure handling, indexing, and caching, are provided in Appendix B and summarized in Table 8.

5 Results

5.1 Scaling to the Full 618M-Token Collection

Table 1 presents the primary scaling results. MEMONDEMAND outperforms LB#1, the strongest published ENTERPRISERAG result, at all seven corpus tiers from 10M tokens through the complete 618M-token collection. The gain is 12.23% at 10M, remains 8.26% at 100M and 6.26% at 250M, and is still 4.66% on the full collection. At 618M, Correct remains 80.0% and Complete reaches 74.28%, despite Document Recall falling to 48.77% and InvDoc increasing to 0.5019. This result exposes the principal scaling trade-off: source recovery and citation validity become harder over the full corpus, yet the retrieved evidence remains sufficiently useful for MEMONDEMAND to preserve a clear Combined advantage. Figure 4 provides the corresponding trends.

5.2 Token Consumption Analysis

To evaluate whether dual memory reduces token consumption, we fix the retrieval trace and vary the representation provided to the answer model. Distilled memory is used for retrieval and ranking in all settings, while detailed memory is loaded according to the tested policy. The token comparison therefore isolates the packing effect of separating compact routing memory from detailed answer memory; the selective-detail quality rows are refreshed with the revised evidence-grounded answer prompt used for the headline runs.

Table 2 shows that dual memory provides a strong quality–cost trade-off. At 10M and 20M source tokens, selective detailed-memory loading reduces answer-input tokens by 63.9% and 70.2% relative to detailed-only loading while reaching Combined scores of 80.45% and 78.66%. Distilled-only loading uses fewer tokens but causes a large quality drop, showing that compact memory alone is insufficient for answering. Loading both representations also increases token use without improving performance. These results confirm that dual memory reduces context cost by using distilled memory for retrieval and loading detailed memory only for the most relevant sources.

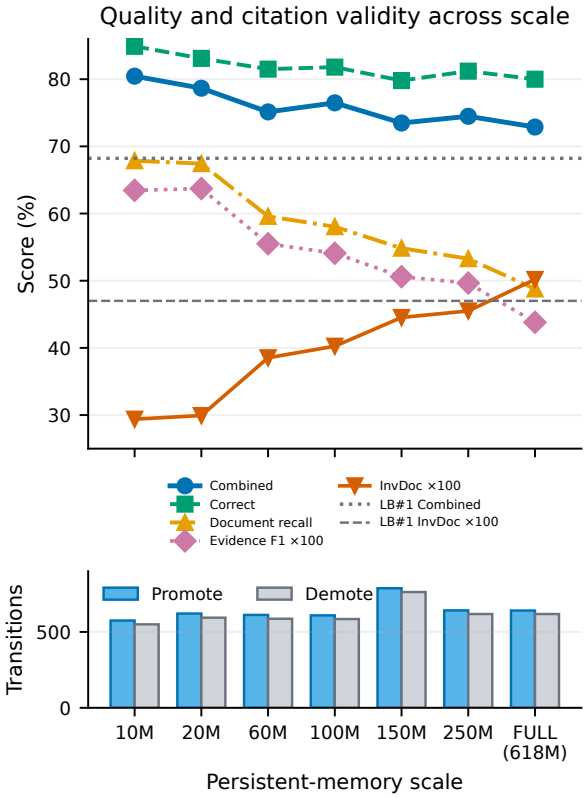

Figure 4: **Quality and managed-state activity across corpus scales.** MEMONDEMAND remains above the published LB#1 Combined reference through the complete 618M-token collection.

Table 14 in Appendix F further decomposes online token use and shows that answer input is the main source of token cost.

5.3 Comparison with Direct Retrieval Strategies

Table 3 compares MEMONDEMAND with four direct retrieval strategies at both 10M and 20M source tokens: BM25 (Robertson and Zaragoza, 2009), flat dense RAG (Karpukhin et al., 2020), hierarchical retrieval with detailed memory only, and direct long-context packing (Liu et al., 2024). All methods use the same 500 questions, answer and evaluation models, and semantic embedding backend; the MEMONDEMAND rows report the revised evidence-grounded answer prompt used in the headline evaluation.

Across both scales, MEMONDEMAND achieves the highest Combined and Correct scores while

Tier	Combined ↑ (%)	Δ Combined	Correct ↑ (%)	Complete ↑ (%)	DocRcl ↑ (%)	Evid. F1 ↑	InvDoc ↓	Promo	Demote
LB#1	**68.22**	–	81.6	72.86	79.02	–	0.4700	–	–
10M	**80.45**	+12.23%	84.9	81.91	67.86	0.6345	0.2939	575	550
20M	**78.66**	+10.44%	83.1	80.08	67.43	0.6372	0.2993	621	594
60M	**75.13**	+6.91%	81.5	76.19	59.58	0.5550	0.3853	612	587
100M	**76.48**	+8.26%	81.8	77.91	58.04	0.5409	0.4025	609	585
150M	**73.48**	+5.26%	79.8	74.93	54.84	0.5060	0.4453	788	763
250M	**74.48**	+6.26%	81.2	76.44	53.28	0.4965	0.4550	642	618
FULL (618M)	**72.88**	+4.66%	80.0	74.28	48.77	0.4381	0.5019	641	618

Table 1: **Primary ENTERPRISERAG scaling results.** LB#1 is the published benchmark reference, Δ Combined is the absolute difference from LB#1, and Promo and Demote count accepted state transitions over 500 queries. FULL is the complete 618M-token collection.

Scale	Memory provided for answering	Answer-input tokens/query	Change (%)	Combined (%) ↑
10M	Detailed only	46,994	–	82.68
10M	Both representations	49,174	+4.6%	70.60
10M	Distilled only	2,963	-93.7%	51.08
10M	**Selective detailed memory (MEMONDEMAND)**	**16,944**	**-63.9%**	**80.45**
20M	Detailed only	45,784	–	77.91
20M	Both representations	51,078	+11.6%	68.22
20M	Distilled only	3,082	-93.3%	49.76
20M	**Selective detailed memory (MEMONDEMAND)**	**13,629**	**-70.2%**	**78.66**

Table 2: **Token consumption under different memory policies at 10M and 20M.** Change is measured relative to detailed-only loading. The selective-detail rows report revised-prompt quality; token counts retain the matched packing traces.

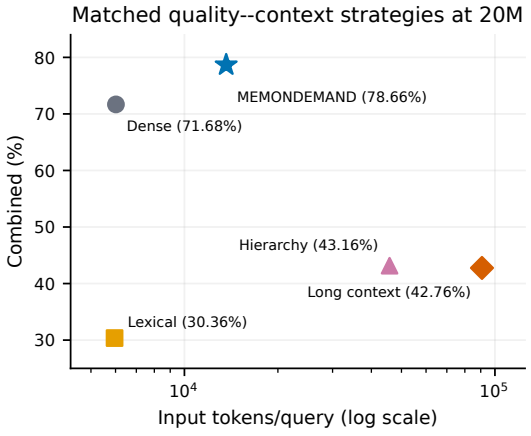

Figure 5: **Quality–context configurations at 20M.** Complete strategies use their own evidence-loading policies; the horizontal axis is logarithmic.

substantially improving Document Recall over flat dense RAG. Although flat dense RAG uses fewer answer-input tokens, its retrieval coverage and answer quality are consistently lower. Detailed-only hierarchy retrieval and direct long-context packing consume several times more context but still produce much lower Combined scores. These results show that MEMONDEMAND provides the strongest quality–cost balance by combining hierarchical routing, selective detailed-memory loading, and on-demand promotion. Figure 5 places the complete retrieval strategies on the quality–context frontier. MEMONDEMAND achieves the strongest Combined score without the high answer-input cost of detailed-only hierarchy retrieval or direct long-context packing.

5.4 Component Ablations

We evaluate the contribution of key design choices by modifying one component at a time. The no-promotion variant removes both promotion and demotion while keeping detailed-memory loading under the same answer-input budget. The one-step variant limits hierarchy navigation depth, and the larger-candidate variant increases the candidate pool from 15 to 25 sources.

We also compare the full system with flat dense retrieval, global source-leaf search without promotion, and the published LB#1 reference. These comparisons separate the effects of hierarchical navigation and adaptive memory updates from di-

Scale	Strategy	Answer-input tokens/query	Combined (%) ↑	Correct (%) ↑	DocRcl (%) ↑
10M	Flat dense RAG	6,001	74.48	74.6	52.91
	BM25 RAG, no promotion	5,726	29.68	34.4	24.43
	Hierarchical, detailed only	46,994	46.80	52.2	70.03
	Direct long-context packing	93,318	44.44	49.4	78.05
	MEMONDEMAND	**16,944**	**80.45**	**84.9**	**67.86**
20M	Flat dense RAG	6,018	71.68	71.8	48.09
	BM25 RAG, no promotion	5,957	30.36	35.0	23.11
	Hierarchical, detailed only	45,784	43.16	48.2	66.87
	Direct long-context packing	90,832	42.76	47.8	75.68
	MEMONDEMAND	**13,629**	**78.66**	**83.1**	**67.43**

Table 3: **End-to-end comparison with direct retrieval strategies at 10M and 20M.** All methods use the same 500 questions, answer/evaluation models, and embedding backend; the MEMONDEMAND rows use the revised answer prompt. MEMONDEMAND achieves the highest Combined and Correct scores at both scales while using substantially fewer answer-input tokens than detailed-only hierarchy retrieval and direct long-context.

rect source-level retrieval.

Table 4 shows that all three components improve Combined. One-step navigation, no promotion, and a 25-candidate pool reduce the score from 80.45% to 64.52%, 64.20%, and 62.64%, showing the value of deeper traversal, adaptive prioritization, and focused selection. The largest drop from the expanded candidate pool further indicates that exposing more sources can introduce distractors rather than improve evidence quality.

The full system also achieves the highest Combined score among all methods. Global source-leaf search reaches similar Document Recall but lower Correct and Combined, while flat dense retrieval and LB#1 achieve only 74.48% and 68.22% Combined. This contrast shows that retrieval coverage alone does not determine answer quality. These results show that MEMONDEMAND benefits from coordinating retrieval depth, promotion, and evidence selection rather than maximizing recall alone.

5.5 Efficiency of On-Demand Promotion

We evaluate whether on-demand promotion reduces the wall time required to prepare reusable parent-level memory. At 20M source tokens, MEMONDEMAND prepares parent representations only when they are promoted by observed query demand, resulting in a measured wall time of 1.46 seconds. In contrast, eagerly preparing all promotable parent representations is estimated to require 593.5 seconds under the measured per-call latency, yielding a $407\times$ reduction in up-front wall time. This avoids spending computation on parent nodes that may never be reused. Table 5 and Figure 11 report the comparison.

5.6 Online Use of Promotion

We examine how often on-demand promotion contributes during online retrieval at 20M source tokens. MEMONDEMAND uses 2.9 hierarchy-navigation steps per query on average, with a 95th-percentile depth of three steps, showing that promotion operates within a bounded retrieval process. Promotion is activated for 470 of the 500 questions, or 94.0% of the evaluation set, indicating that it broadly updates candidate priority and reusable memory state. Together with the no-promotion ablation in Section 5.4, these results show that promotion is both frequently used during retrieval and important for final performance. Table 6 reports the detailed activity statistics, and Figure 12 visualizes the corresponding promotion profile.

5.7 External Benchmark Evaluation

We evaluate MEMONDEMAND on three external benchmarks with different retrieval and reasoning requirements. FinanceBench tests financial-document question answering (Islam et al., 2023), HotpotQA tests multi-hop reasoning across documents (Yang et al., 2018), and FRAMES tests fact retrieval with additional reasoning (Krishna et al., 2025). As shown in Table 7, MEMONDEMAND achieves strong results across all three settings, including 80.00% Document Recall on FinanceBench, 78.4% Correct on HotpotQA, and 88.76% Document Recall on FRAMES. These results show that the same memory design remains effective across financial, multi-hop, and fact-retrieval tasks rather than being limited to EnterpriseRAG-Bench.

The results confirm that MEMONDEMAND generalizes across datasets with different document

Method	DocRcl (%) ↑	Correct (%) ↑	Combined (%) ↑	Δ vs. Full (%)
System variants				
Full system	67.86	84.9	80.45	–
One-step navigation	59.41	79.4	64.52	−15.93%
No promotion	59.60	82.7	64.20	−16.25%
25-candidate pool	57.44	78.4	62.64	−17.81%
Retrieval baselines				
Flat dense retrieval	52.91	74.6	74.48	−5.97%
Global source-leaf search, no promotion	68.25	81.2	76.68	−3.77%
Published reference				
LB#1	79.02	81.6	68.22	−12.23%

Table 4: **Component ablations and retrieval comparisons at 10M.** System variants modify one design while keeping the remaining settings fixed. Δ reports the absolute Combined difference from the full system.

Promotion strategy	Wall time
On-demand promotion	1.46 s
Eager promotion of all parent nodes	593.5 s
Wall-time ratio	407×

Table 5: **Preparation wall time under different promotion strategies at 20M.** On-demand promotion avoids eagerly preparing every promotable parent node.

Online quantity	Value
Mean navigation steps per query	2.9
95th-percentile navigation steps	3
Questions activating promotion	470 / 500 (94.0%)

Table 6: **Promotion activity at 20M.** Statistics cover the complete set of 500 evaluation queries.

structures, evidence patterns, and reasoning demands. Its dynamic hierarchy, dual memory, and on-demand promotion remain effective beyond the primary enterprise benchmark.

6 Conclusion

We present MEMONDEMAND, a memory management system for large-scale enterprise data. It combines a dynamic hierarchy, dual memory, and on-demand promotion. Across EnterpriseRAG-Bench scales from 10M tokens through the complete 618M-token collection, MEMONDEMAND consistently outperforms the published LB#1 result. Results on external benchmarks further show that the same design transfers across different settings. **Overall, MEMONDEMAND provides a practical solution for extending LLMs with ultra-large memory beyond the limits of the model context window**.

Limitations

Our main scaling results and promotion-floor sweeps are based on single runs, so repeated-run variance and confidence intervals remain to be studied. The eager preparation cost is estimated from measured call latency rather than measured through a complete end-to-end run. In addition, Document Recall decreases at the largest scale, and the external benchmarks show that strong source recovery does not always translate into equally strong answer correctness. Future work should evaluate more domains, longer query streams, and broader model and retrieval configurations.

References

Daniel J. Abadi, Daniel S. Myers, David J. DeWitt, and Samuel Madden. 2007. Materialization strategies in a column-oriented DBMS. In *Proceedings of the 23rd International Conference on Data Engineering*, pages 466–475. IEEE.

Michael Armbrust, Ali Ghodsi, Reynold Xin, and Matei Zaharia. 2021. Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. In *Conference on Innovative Data Systems Research*.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2006. Bigtable: A distributed storage system for structured data. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, pages 205–218.

Surajit Chaudhuri and Vivek Narasayya. 2007. Self-tuning database systems: A decade of progress. *Pro-*

Benchmark	Questions	DocRcl (%) ↑	Evid. F1 ↑	Correct (%) ↑	InvDoc ↓
FinanceBench	150	80.00	0.729	66.0	0.289
HotpotQA	500	49.90	0.549	78.4	0.320
FRAMES	500	88.76	0.617	55.0	0.439

Table 7: **External benchmark results.** MEMONDEMAND maintains strong retrieval and answering performance across financial-document, multi-hop, and fact-retrieval.

ceedings of the 33rd International Conference on Very Large Data Bases, pages 3–14.

Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. SPANN: Highly-efficient billion-scale approximate nearest neighbor search. In *Advances in Neural Information Processing Systems*, volume 34, pages 5199–5212.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.

Edgar F. Codd. 1970. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. RULER: What's the real context size of your long-context language models? In *Conference on Language Modeling*.

Stratos Idreos, Martin L. Kersten, and Stefan Manegold. 2007. Database cracking. In *Proceedings of the Conference on Innovative Data Systems Research*, pages 68–78.

Piotr Indyk and Haike Xu. 2023. Worst-case performance of popular approximate nearest neighbor search implementations: Guarantees and limitations. In *Advances in Neural Information Processing Systems*, volume 36.

Pranab Islam, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. 2023. FinanceBench: A new benchmark for financial question answering. *arXiv preprint arXiv:2311.11944*.

Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. DiskANN: Fast accurate billion-point nearest neighbor search on a single node. In *Advances in Neural Information Processing Systems*, volume 32.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. 2024. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7036–7050.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 1658–1677.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory OS of AI agent. *arXiv preprint arXiv:2506.06326*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 6769–6781.

Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. 2025. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4745–4759.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.

Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, et al. 2025. MemOS: An operating system for memory-augmented generation (MAG) in large language models. *arXiv preprint arXiv:2505.22101*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Yu. A. Malkov and D. A. Yashunin. 2020. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836.

Nimrod Megiddo and Dharmendra S. Modha. 2003. ARC: A self-tuning, low overhead replacement cache. *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 115–130.

Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). *Acta Informatica*, 33(4):351–385.

OpenAI. 2026a. GPT-5.4 mini model. `https://developers.openai.com/api/docs/models/gpt-5.4-mini`. Accessed 2026-07-30.

OpenAI. 2026b. GPT-5.4 model. `https://developers.openai.com/api/docs/models/gpt-5.4`. Accessed 2026-07-30.

OpenAI. 2026c. text-embedding-3-small model. `https://developers.openai.com/api/docs/models/text-embedding-3-small`. Accessed 2026-07-30.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22. ACM.

Hongjin Qian, Peitian Zhang, Zheng Liu, Kelong Mao, and Zhicheng Dou. 2024. MemoRAG: Moving towards next-generation retrieval-augmented generation via memory-inspired knowledge discovery. *arXiv preprint arXiv:2409.05591*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 3982–3992.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations*.

Aayush D. Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. 2021. Freshdiskann: A fast and accurate graph-based ANN index for streaming similarity search. *arXiv preprint arXiv:2105.09613*.

Haoran Sun and Shaoning Zeng. 2025. H-MEM: Hierarchical memory for high-efficiency long-term reasoning in large language model agents. *arXiv preprint arXiv:2507.22925*.

Yuhong Sun, Joachim Rahmfeld, Chris Weaver, Roshan Desai, Wenxi Huang, and Mark H. Butler. 2026. EnterpriseRAG-Bench: A RAG benchmark for company internal knowledge. *arXiv preprint arXiv:2605.05253*.

Jianguo Wang, Xiaomeng Yi, Rundong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2614–2627.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. 2023. Augmenting language models with long-term memory. *arXiv preprint arXiv:2306.07174*.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024. RECOMP: Improving retrieval-augmented language models with compression and selective augmentation. In *International Conference on Learning Representations*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2023. MemoryBank: Enhancing large language models with long-term memory. *arXiv preprint arXiv:2305.10250*.

A Related Work

A.1 Retrieval and Hierarchical Organization

Retrieval-augmented generation combines non-parametric search with an answer model (Lewis et al., 2020), while dense passage retrieval learns a shared representation space for queries and passages (Karpukhin et al., 2020). Large-scale vector systems such as HNSW, DiskANN, FreshDiskANN, SPANN, and Milvus improve candidate access through graph-based search, disk indexing, hybrid-memory designs, and distributed execution (Malkov and Yashunin, 2020; Jayaram Subramanya et al., 2019; Singh et al., 2021; Chen et al., 2021; Wang et al., 2021). These methods focus mainly on efficient retrieval from large candidate spaces. In contrast, MEMONDEMAND manages how enterprise memory is organized, represented, loaded, and updated above the underlying search infrastructure. Adaptive-RAG changes retrieval effort according to question complexity, while Self-RAG learns when to retrieve and how to assess retrieved evidence (Jeong et al., 2024; Asai et al., 2024). Their control is mainly query specific, whereas MEMONDEMAND also manages hierarchy construction, dual-memory representation, and cross-query memory updates.

Hierarchical retrieval methods organize information at multiple abstraction levels. RAPTOR recursively clusters and summarizes text into a retrieval tree (Sarthi et al., 2024), GraphRAG constructs community summaries for graph-based retrieval (Edge et al., 2024), and HippoRAG uses graph structure to support long-term associative retrieval (Gutiérrez et al., 2024). H-MEM performs layerwise memory access, while MemoRAG uses compact global memory to guide retrieval from a larger collection (Sun and Zeng, 2025; Qian et al., 2024). These methods show the value of hierarchical abstraction, but they generally rely on a predefined hierarchy construction process. MEMONDEMAND instead determines the hierarchy depth and abstraction structure separately for each collection, while preserving direct access to L0 source nodes when higher-level records omit relevant information.

A.2 Compression and Persistent Memory

Long-context models do not use information uniformly across context positions (Liu et al., 2024), and their effective task capacity can be lower than their stated context length (Hsieh et al., 2024). LLMLingua and LongLLMLingua reduce prompt length under a token budget (Jiang et al., 2023, 2024), while RECOMP compresses retrieved documents and can remove unhelpful retrieved content (Xu et al., 2024). These methods reduce the size of a prompt or retrieved batch, but they do not maintain separate retrieval and answering representations at every hierarchy level. MEMONDEMAND stores distilled memory for routing and detailed memory for answering, allowing compact retrieval without replacing the information used for generation and citation.

Persistent-memory systems study how language-model agents store and reuse information across interactions. Generative Agents derives higher-level reflections from past observations (Park et al., 2023), MemoryBank supports long-term updating and forgetting (Zhong et al., 2023), MemGPT manages limited context through a virtual-memory design (Packer et al., 2023), and LongMem separates long-term storage from the answer model (Wang et al., 2023). Mem0, A-MEM, MemoryOS, and MemOS further study memory extraction, organization, updating, and lifecycle control (Chhikara et al., 2025; Xu et al., 2025; Kang et al., 2025; Li et al., 2025). Most of these systems focus on interaction history, user memory, or agent experience. MEMONDEMAND instead targets very large enterprise collections and jointly supports dynamic hierarchy construction, dual memory at each level, selective detailed-memory loading, and on-demand promotion under bounded active state.

A.3 Database Physical Design

Database systems preserve stable record access while allowing storage layouts, materialized views, and indexing structures to change (Codd, 1970). Log-structured storage supports continuous updates, database cracking adapts access paths to observed queries, deferred materialization delays expensive data access, adaptive replacement manages limited cache capacity, and self-tuning systems revise physical design as workloads change (O'Neil et al., 1996; Idreos et al., 2007; Abadi et al., 2007; Megiddo and Modha, 2003; Chaudhuri and Narasayya, 2007). MEMONDEMAND brings these principles to enterprise memory by constructing a hierarchy for each collection, using distilled memory for efficient retrieval, loading detailed memory only when required for answering, and updating node priority through promotion and demotion as query demand changes.

B Implementation Details

B.1 Model and retrieval configuration.

GPT-5.4 mini constructs L0 distilled memory and extracts key facts, while GPT-5.4 performs higher-level hierarchy construction, navigation, source selection, answer generation, and ENTERPRISERAG evaluation. All semantic indices use 1,536-dimensional `text-embedding-3-small` embeddings (OpenAI, 2026a,b,c). Table 8 summarizes the model assignment for each operation.

Evaluation protocol. We follow the ENTERPRISERAG definitions for answers, supporting evidence, and source IDs (Sun et al., 2026). Each generated answer is compared with the canonical answer, and each returned source ID is checked against the benchmark evidence set. The predefined `STOP_INSUFFICIENT` rule is applied during aggregation.

Operation	API identifier
L0 distillation and key facts	`gpt-5.4-mini`
Higher-level decision and abstraction	`gpt-5.4`
Navigation and source selection	`gpt-5.4`
Answer generation	`gpt-5.4`
ENTERPRISERAG evaluation	`gpt-5.4`
Semantic indexing	`text-embedding-3-small`

Table 8: **API assignment.** GPT-5.4 mini is used for L0 memory construction, while GPT-5.4 handles higher-level construction(L1 L2 and L3), retrieval, answering, and evaluation.

B.2 Hierarchy Footprint

Table 9 reports the realized hierarchy size at each corpus tier, including the complete 618M-token collection. The number of L0 nodes increases with corpus scale, while the dynamic construction procedure determines the number of L1 and L2 nodes for each collection.

Tier	L0 nodes	L1 nodes	L2 nodes
10M	8,927	94	9
20M	17,051	287	9
60M	49,978	223	14
100M	82,779	287	16
150M	124,330	352	18
250M	207,179	455	21
FULL (618M)	511,962	716	27

Table 9: **Realized hierarchy sizes.** The dynamic construction procedure determines the number of nodes at each level for every corpus tier.

C Promotion-Floor Sensitivity

We control promotion through a similarity floor θ: a candidate node v is promoted only when its promotion score satisfies

$$g_t(v) \geq \theta,$$

where $g_t(v) \in [0, 1]$ measures its similarity to the current query and usage state. If the floor is too low, many weakly related nodes are promoted, which introduces noisy priority signals and can displace more useful candidates. If the floor is too high, few nodes are promoted, limiting the system's ability to adapt retrieval priority across queries. Tables 10, 11, and 12, together with Figures 6, 7, and 8, show the same non-monotone pattern at 10M, 150M, and the complete 618M-token collection: performance is lower under either excessive or insufficient promotion and is highest at an intermediate level. A floor of 0.5 achieves the best Combined score in every sweep, reaching 76.40% at 10M, 71.91% at 150M, and 72.88% on the complete collection. At 618M, a floor of 0.6 is close in quality at 72.86% but permits only 332 promotions, whereas the global optimum at 0.5 records 641 promotions and 618 demotions. We therefore use $\theta = 0.5$ as the default promotion floor. The 10M and 150M sweeps are matched sensitivity runs distinct from the revised-prompt results in Table 1; the 618M sweep uses the revised prompt, and its 0.5 setting is the FULL result in the primary table.

Floor	Combined (%) ↑	Promo
0.1	72.65	717
0.2	68.40	682
0.3	70.61	708
0.4	70.00	700
0.5	**76.40**	**595**
0.6	72.80	283
0.7	68.80	66
0.8	68.80	19
0.9	70.20	2

Table 10: **Promotion-floor sensitivity at 10M.** Combined follows an inverted-U trend and is highest at a floor of 0.5. Promo counts accepted promotion decisions.

D Detailed Algorithms

This appendix provides the full procedures for the three mechanisms introduced in Section 3: dynamic hierarchy construction, dual-memory retrieval with selective detailed-memory loading, and on-demand promotion with demotion.

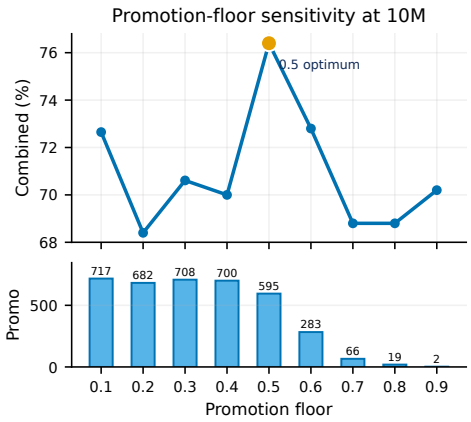

Figure 6: **Promotion-floor sensitivity at 10M.** Performance peaks at an intermediate similarity floor, while promotion activity decreases as the floor increases.

Floor	Combined (%) ↑	Promo
0.1	69.21	799
0.2	71.36	797
0.3	71.03	802
0.4	71.34	614
0.5	**71.91**	**774**
0.6	70.55	302
0.7	69.80	73
0.8	69.98	4
0.9	71.46	0

Table 11: **Promotion-floor sensitivity at 150M.** The same inverted-U pattern appears at the larger scale, with the highest Combined score at a floor of 0.5.

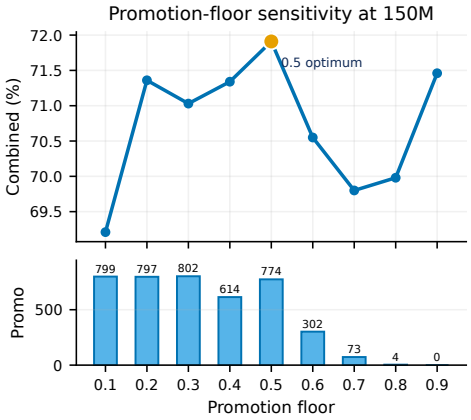

Figure 7: **Promotion-floor sensitivity at 150M.** An intermediate similarity floor again provides the best balance between excessive and insufficient promotion.

Dynamic hierarchy construction. Algorithm 2 starts from the L0 source nodes and iteratively proposes a higher abstraction level. At each step, the controller evaluates the proposed grouping and either accepts it, requests a revised grouping, or stops construction. The process therefore determines both the structure and depth of the hierarchy from

Floor	Combined (%) ↑	Promo	Demote
0.1	71.81	795	773
0.2	71.09	792	770
0.3	70.95	808	786
0.4	72.36	793	770
0.5	**72.88**	**641**	**618**
0.6	72.86	332	320
0.7	72.11	77	74
0.8	70.67	5	5
0.9	70.82	0	0

Table 12: **Promotion-floor sensitivity on FULL EnterpriseRAG (618M tokens).** A floor of 0.5 attains the highest Combined score in the nine-point sweep. Promo and Demote count accepted state transitions across 500 queries.

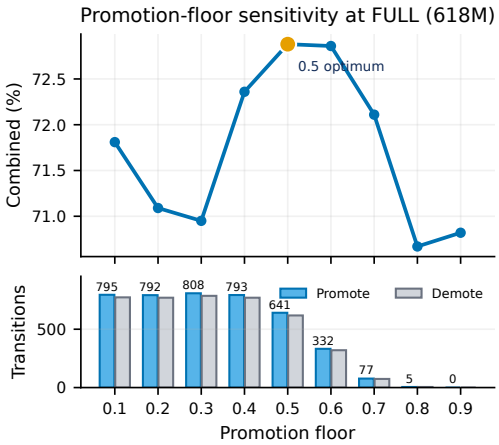

Figure 8: **Promotion-floor sensitivity on the complete 618M-token collection.** Quality peaks at 0.5; both promotion and demotion activity fall sharply as the floor becomes more selective.

the current collection rather than fixing them in advance.

Dual-memory retrieval and selective detailed-memory loading. Algorithm 3 performs coarse-to-fine retrieval over distilled memory until it reaches L0. Signals from different hierarchy paths are merged for each source candidate, after which only the highest-ranked detailed memories that fit the answer-input budget are loaded. This procedure uses compact memory for retrieval while reserving the answer context for selected detailed evidence.

On-demand promotion and demotion. Algorithm 4 updates node priority from query-time evidence and prior usage. Nodes above the promotion threshold receive a current-query score increase and enter the cross-query active state. Repeated use refreshes their retention, while expired or low-retention nodes are demoted when the active-state

Algorithm 2 Dynamic multi-level hierarchy construction

Require: L0 nodes $\mathcal{V}_0$, maximum depth D, maximum proposals A
$\mathcal{T} \leftarrow \mathcal{V}_0$; $\ell \leftarrow 0$; $a \leftarrow 0$
while $\ell < D$ and $a < A$ **do**
  $\mathcal{C} \leftarrow \textsc{ProposeGroups}(\text{dist}(\mathcal{V}_\ell))$; $a \leftarrow a + 1$
  $s \leftarrow \textsc{ProposalStats}(\mathcal{C})$
  $u \leftarrow \textsc{Controller}(s)$
  if $u = \textsc{Stop}$ **then**
    break
  end if
  if $u = \textsc{Recluster}$ **then**
    revise the grouping proposal
    continue
  end if
  $\mathcal{V}_{\ell+1} \leftarrow \textsc{BuildParentRecords}(\mathcal{C})$
  for $p \in \mathcal{V}_{\ell+1}$ **do**
    $\text{children}(p) \leftarrow \{v \in \mathcal{V}_\ell : v \text{ belongs to } p\}$
  end for
  add $\mathcal{V}_{\ell+1}$ and their edges to $\mathcal{T}$
  $\ell \leftarrow \ell + 1$
end while
return $\mathcal{T}$

Algorithm 3 Dual-memory retrieval and selective detailed-memory loading

Require: query q, hierarchy $\mathcal{T}$, answer-input budget B
$\mathcal{V} \leftarrow \textsc{HighestLevel}(\mathcal{T})$
while $\mathcal{V} \not\subseteq L_0$ **do**
  $\mathcal{S} \leftarrow \textsc{RetrieveDistilled}(q, \mathcal{V})$
  $\mathcal{V} \leftarrow \bigcup_{v \in \mathcal{S}} \text{children}(v)$
end while
$\mathcal{R} \leftarrow \textsc{MergeRouteSignals}(\mathcal{V})$
$\mathcal{C} \leftarrow \textsc{RankCandidates}(\mathcal{R})$
$X \leftarrow \varnothing$; $t_{\text{used}} \leftarrow \textsc{FixedAnswerCost}(q)$
for $v \in \mathcal{C}$ **do**
  if $t_{\text{used}} + \tau_v^d \le B$ **then**
    append d_v to X
    $t_{\text{used}} \leftarrow t_{\text{used}} + \tau_v^d$
  end if
end for
$(a, C_{\text{pred}}) \leftarrow \textsc{Answer}(q, X)$
$C_{\text{pred}} \leftarrow C_{\text{pred}} \cap \text{ID}(X)$
return (a, C_{pred}, X)

Algorithm 4 On-demand memory promotion and demotion

Require: query q_t, candidates $\mathcal{C}_t$, active state $\mathbf{z}_t$, threshold θ, active-state budget K
for $v \in \textsc{Deduplicate}(\mathcal{C}_t)$ **do**
  $g_t(v) \leftarrow \textsc{PromotionScore}(q_t, v, \mathbf{z}_t)$
  if $g_t(v) \ge \theta$ **then**
    increase the current ranking score of v
    upsert $(v, g_t(v), t)$ into $\mathbf{z}_t$
    append $\textsc{Promote}(v)$ to the transition log
  else if $v \in \text{active}(\mathbf{z}_t)$ **then**
    refresh the score and last-use time of v
  end if
end for
for $v \in \text{active}(\mathbf{z}_t)$ **do**
  $\rho_t(v) \leftarrow \max\left(0, s_t(v) - \frac{t - t_{\text{last}(v)}}{\tau}\right)$ (Equation 2)
end for
while $|\,\text{active}(\mathbf{z}_t)| > K$ or an active node has expired **do**
  demote the expired or minimum-retention node
  append $\textsc{Demote}$ to the transition log
end while
return updated ranking scores and $\mathbf{z}_{t+1}$

budget is exceeded.

E System Properties

This section states several properties of the MEMONDEMAND execution process. These results describe retrieval coverage, candidate use, token accounting, and deferred preparation; they do not guarantee answer correctness.

Complementary retrieval routes. Let H_d denote the event that a gold source is recovered through direct L0 retrieval, and let H_h denote recovery through hierarchical retrieval over distilled memory. The recall of their union is

$$R_{\text{fuse}} = \Pr(H_d \cup H_h) = R_d + \Pr(H_h \cap H_d^c).$$

The hierarchical route therefore improves recall only when it recovers a source missed by direct L0 retrieval. This result explains why MEMONDEMAND combines the two routes rather than replacing direct retrieval with hierarchy traversal.

Let $L = (h_1, h_2, \ldots)$ be a ranked list of retrieval hits, and let $\sigma(h)$ denote the L0 source associated with hit h. We compare two procedures under a source budget k: taking the first k hits and then removing duplicates, or scanning L until k distinct sources are collected.

Proposition 1 (Distinct-source coverage)**.** *If L contains at least k distinct sources, scanning until k distinct sources are collected returns a superset of the sources obtained by deduplicating the first k hits.*

Proof. Every source appearing among the first k hits is encountered before the distinct-source scan terminates. Duplicate hits allow the scan to continue to later positions and include additional sources. Therefore, distinct-source collection cannot reduce source coverage under the same budget. □

Answer-input accounting. Let E_q be the set of L0 detailed memories loaded for query q, and let $T_0(q)$ denote the fixed token cost of the question and instructions. The total answer-input cost is

$$T(q) = T_0(q) + \sum_{v \in E_q} \tau_v^d \le B.$$

If a detailed-only policy would load a candidate set $\mathcal{C}_q$, selective detailed-memory loading saves $\sum_{(v \in \mathcal{C}_q \setminus E_q)} \tau_v^d$ tokens. Distilled-memory tokens are used during retrieval and do not enter the answer-input budget.

Deferred-preparation cost. Let c_v be the cost of preparing a representation for node v, and let T_v be the first query step at which that representation is needed. Over a query horizon H, the expected on-demand preparation cost is:

$$\begin{aligned} \mathbb{E}[C_{\text{on-demand}}(H)] &= \sum_v c_v \Pr(T_v \le H) \\ &\le \sum_v c_v = C_{\text{eager}}. \end{aligned} \tag{3}$$

Thus, preparing a representation only when it is first needed cannot cost more than preparing every representation in advance over the same horizon. This is a cost-accounting result rather than an end-to-end latency guarantee, and it applies only to representations whose construction is actually deferred.

F Additional Scaling and Component Results

F.1 Scaling of Direct Source-Level Retrieval

Table 13 reports the performance of global source-leaf search without promotion across increasing collection sizes. Document Recall and Correct generally decrease as the collection grows, showing the limits of direct source-level retrieval without adaptive hierarchy navigation and promotion.

Scale	DocRcl (%) ↑	Correct (%) ↑
10M	68.25	81.2
20M	65.19	80.2
60M	55.02	74.2
100M	59.04	74.4
150M	55.30	77.2

Table 13: **Global source-leaf retrieval across corpus scales.** Document Recall and Correct are reported as percentages.

F.2 Online Token Use

Table 14 decomposes online token use at 20M into retrieval-control and answer-generation costs. Mean is the average number of tokens per query, Median is the 50th-percentile value, and p95 is the 95th-percentile value, meaning that 95% of queries use no more than this number of tokens. Answer input is the largest component, while retrieval-control output and answer output account for only a small share of the total.

Runtime component	Mean	Median	p95
Retrieval-control input	3,758	3,969	4,270
Retrieval-control output	155	155	186
Answer input	13,629	13,284	19,413
Answer output	142	124	291
Total online tokens	17,683	17,425	23,534

Table 14: **Online token use at 20M.** Columns report the mean, median, and 95th-percentile token count per query. Total tokens include retrieval-control input and output and answer input and output.

F.3 External visualization Results

Figure 9 compares answer quality with answer-input cost across evidence-loading policies. It shows that selective detailed-memory loading preserves strong Combined performance while using substantially fewer tokens than detailed-only or combined-representation loading.

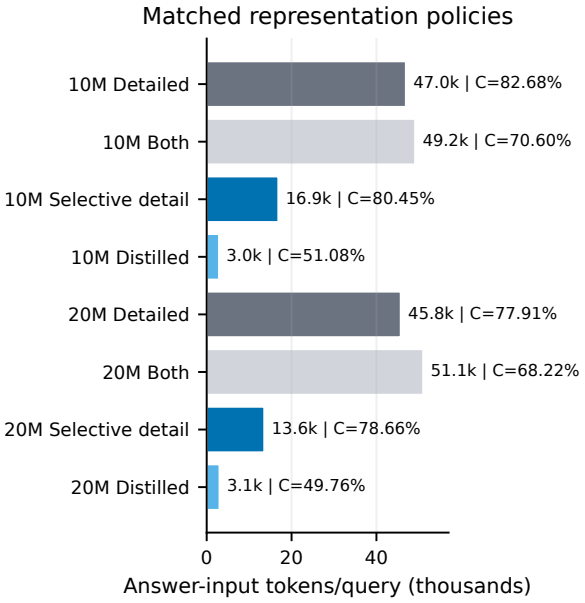

Figure 9: **Quality and answer-input cost.** Labels report Combined for the evidence-loading policies in Table 2.

Figure 10 visualizes the degradation caused by removing or restricting individual components. The full system performs best, confirming that multi-step navigation, on-demand promotion, and controlled candidate selection contribute jointly to final quality.

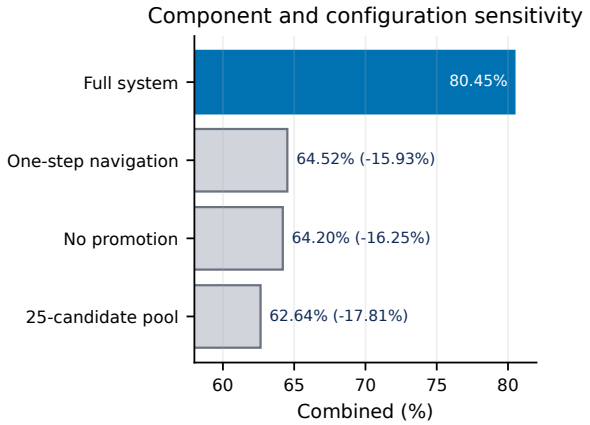

Figure 10: **Component and configuration sensitivity.** Promotion removal disables both the current-query bonus and the managed-state update.

Figure 11 compares the up-front wall time of on-demand promotion with eager preparation of all parent-level representations. The large gap shows that on-demand promotion avoids preparing memory that may never be reused, substantially reducing the cost required before query execution.

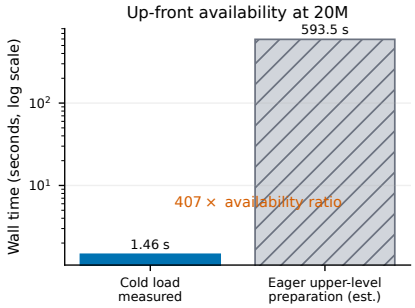

Figure 11: **Up-front availability at 20M.** The cold load is measured, while eager parent preparation is estimated.

Figure 12 summarizes retrieval and answering performance on FinanceBench, HotpotQA, and FRAMES. The results show that source recovery and answer correctness vary across tasks, reflecting their different evidence structures and reasoning requirements.

G Prompt Templates

This section provides the prompt templates used for memory construction, hierarchy control, retrieval, promotion, answer generation, and evaluation. The final answer template is transcribed from `improved_answer_prompt.py`, with only TeX escaping and line wrapping changed for presentation.

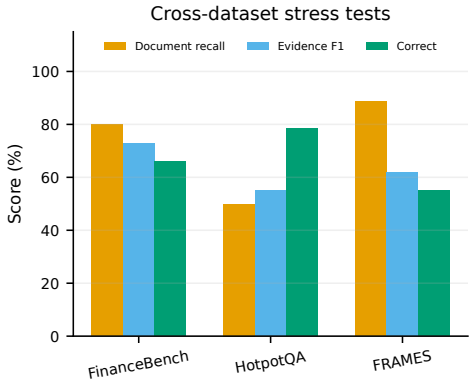

Figure 12: **External benchmark profile.** Source recovery and answer correctness vary across the three evaluation settings.

Source-leaf distilled-record construction

System: Rewrite the source into a distilled routing record containing topic, entities, dates, identifiers, and decision-bearing facts. Preserve negation and qualifiers. Do not add outside knowledge. Return JSON with `distilled_text`, `key_facts`, and `source_id`.
User: One source chunk with tenant metadata, section path, and source ID.

Parent-abstraction decision

System: Inspect coherence, residual noise, group sizes, representative content, and remaining budget. Return exactly `CREATE`, `RECLUSTER`, or `STOP`, plus a reason. Never alter source IDs or descendant links.
User: Proposal statistics and representative distilled routing records.

Parent-abstraction construction

System: Summarize the child records for routing. Preserve entities, time ranges, constraints, exceptions, and disagreements. Return the unchanged descendant source IDs and introduce no facts absent from the children.
User: A bounded cluster of distilled routing records and source IDs.

Navigation and source selection

System: Use distilled routing records only for search. Return source IDs from the supplied candidate set and stop when the evidence need is resolved or the navigation budget is spent.
User: Question, hierarchy frontier, route scores, and managed-state metadata.

Promotion gate

System: Return each supplied source ID, promotion decision, and score. Promotion may change current priority and later reuse but does not load a detailed evidence payload.
User: Question, unique-source candidates, route signals, state, and capacity.

Evidence-grounded answer

System: You are an enterprise memory question-answering assistant.
You will be given: (1) a user `QUERY`; (2) `[CTX]` hierarchy context blocks for background scope, which must not be cited; and (3) `[EVID]` evidence blocks, each with a `node_id`. Some evidence blocks contain only a `[BRIEF]` distilled summary; top-ranked evidence also contains the complete `[FULL]` source text.
Critical anti-hallucination rules:

- Every specific value stated—a number, threshold, date, flag name, configuration key, dollar amount, percentage, duration, person name, or other concrete detail—must be an exact copy of text that literally appears in an `[EVID]` block. Do not paraphrase numbers, infer a value, or estimate one even if it appears reasonable.
- Before writing any specific value, silently verify: "Can I point to the exact substring in `[EVID]` that contains this value?" If not, omit the value or state that the document does not specify it.
- If part of the answer is supported but another part requires guessing a specific value, answer only the supported part and explicitly note what is not specified.
- Never fill in a plausible number, date, or threshold merely to be helpful. An incomplete but accurate answer is preferable to a complete but partially fabricated answer.
- Inspect conversational or reply sections in `[FULL]` text, including review conversations, comment threads, and follow-up messages. These may contain the controlling configuration key or value rather than the initial description.

Rules:

- Write a precise, factual answer of one to five sentences using only information from `[EVID]` blocks.
- Ground the answer primarily in `[FULL]` source text. Treat `[BRIEF]` as a topic hint only. Write one to three precise sentences, and make every factual claim traceable to a specific `[FULL]` passage.
- After the answer, on a new line, output `CITED: id1,id2,...`.
- Cite every `[EVID]` node whose `[FULL]` or `[BRIEF]` content was used. If three or more nodes contributed, cite all of them. Do not cite `[CTX]` node IDs.
- If no `[EVID]` context applies, output `INSUFFICIENT EVIDENCE` and an empty `CITED:` line. Never invent facts absent from the provided context.

User: Query, hierarchy context blocks, and evidence blocks with source IDs.

Benchmark evaluation

System: Compare the candidate and canonical answers. Return schema-constrained JSON with `correct`, `validated_facts`, and `missing_facts`.
User: Question, candidate answer, and canonical answer.