APDMem: Agent-Controlled Progressive Disclosure for Query-Adaptive Long-Term Memory

Chin-Lun Fu Anagha Kulkarni Hong Ni Behrouz Madahian
Machine Learning Center of Excellence, JPMorgan Chase & Co.
chin-lun.fu@jpmchase.com, anagha.p.kulkarni@jpmorgan.com
hong.ni@jpmchase.com, behrouz.madahian@jpmchase.com

Abstract

Personalized LLM assistants must recover sparse evidence from long conversation histories across queries of varying complexity. We introduce **APDMem** (**A**gent-controlled **P**rogressive **D**isclosure **Mem**ory), a hierarchical long-term memory architecture that applies progressive disclosure to memory retrieval. Rather than relying on a flat memory store or fixed retrieval granularity, APDMem represents conversation history as four progressively detailed layers: thematic summaries, personalized key facts, turn-level evidence notes, and raw messages. At inference time, a *controller* applies progressive disclosure to the memory hierarchy: it first reads high-level summaries and drills into finer evidence only when needed. This creates an adaptive cost–fidelity trade-off: simple queries can terminate early, while complex temporal, multi-hop, or exact-evidence queries trigger deeper inspection. A *note synthesizer* converts retrieved evidence into a query-focused structure that consolidates facts, orders events, and flags contradictions before final answer generation. Experiments on LongMemEval show that APDMem achieves strong performance for long-context memory reasoning while accessing only 8% of the total conversations.

1 Introduction

Large Language Model (LLM) agents have shown strong performance on complex, multi-step tasks. However, personalized agents must operate across long-term, multi-session interactions, where relevant information may be scattered across many past conversations (Xu et al., 2026; Liu et al., 2025; Yu et al., 2026). Such agents must remember user preferences, factual details, temporal sequences, and recurring themes over time (Chhikara et al., 2025; Xu et al., 2025; Fang et al., 2026; Liu et al., 2026; Rasmussen et al., 2025). Without effective memory, agents suffer from contextual amnesia: they may fail to recall prior facts, lose track of evolving goals, or overlook cross-session connections.

Prior works on memory systems improve long-term recall by transforming conversation history into specialized representations: MemoryBank uses a structured memory store for personalized dialogue (Zhong et al., 2024); MemGPT manages information via virtual context and external memory (Packer et al., 2024); A-MEM builds interconnected, evolving memory networks (Xu et al., 2025); and Zep constructs temporal, entity-centric memory graphs (Rasmussen et al., 2025). Other systems reduce cost through compression or consolidation: LightMem uses staged memory processing (Fang et al., 2026), while SimpleMem retrieves over compact semantic memories (Liu et al., 2026). While these designs reduce retrieval cost, they also introduce additional memory machinery and require committing conversational context to external memory stores, graph nodes, or other abstractions.

We introduce **APDMem** (**A**gent-controlled **P**rogressive **D**isclosure **Mem**ory), a hierarchical memory system for efficient context recall in personalized agents. APDMem treats long-term memory access as a query-adaptive disclosure problem rather than a one-shot retrieval problem. Inspired by **progressive disclosure** (Nielsen, 1993), it exposes coarse information first and reveals finer evidence as needed. APDMem stores conversations as lightweight Markdown layers—thematic summaries (L_0), personalized key facts (L_1), turn-level evidence notes (L_2), raw messages (L_3)—rather than converting them into a specialized database, graph, or compressed-only representation.

Unlike existing systems that fix the retrieved representation in advance, APDMem only materializes thematic summaries at ingestion time while preserving raw messages underneath. At inference time, the *Progressive Disclosure Controller* expands on demand into finer details like personalized key facts, turn-level evidence notes, or raw

EMNLP 2026 Industry Track, pages 1361–1376
October 24-29, 2026 ©2026 Association for Computational Linguistics

messages. After sufficient evidence is collected, the controller drafts an answer. For final answer generation, *Organized Notes Synthesizer* structures the evidence retrieved by the controller along with the draft answer into organized notes. These notes consolidate relevant evidence, align it with the query and provide the answer generator with structured grounding context instead of raw retrieved snippets, reducing noise and improving answer accuracy.

APDMem frames long-term memory retrieval as a query-adaptive disclosure problem: it must determine *which* conversations matter and *how much detail* to expose. Concretely, APDMem introduces two coupled dimensions: (1) hierarchical memory organization (coarse summaries linked to finer-grained evidence) and (2) progressive disclosure as an explicit architectural principle governing how that hierarchy is accessed. That is, the agent is designed to open only the minimum memory level required, exposing coarse views first and revealing finer evidence only when needed, with the disclosure depth controlled through tool-mediated interactions. Our primary contributions are:

- **Coarse-to-fine memory formulation.** We introduce a *Multi-Resolution Memory Substrate* that preserves raw messages beneath progressively abstract memory layers.
- **Agent-controlled query-adaptive memory disclosure.** We introduce a *Progressive Disclosure Controller* that decomposes queries, handles temporal constraints, and drills into deeper memory layers when needed.
- **Organized Notes for grounded generation.** We introduce an *Organized Notes Synthesizer* that grounds retrieved evidence and the controller's draft answer into structured notes.
- **LongMemEval validation.** We show that APDMem achieves strong performance on LongMemEval (Wu et al., 2025) while accessing only 8% of the conversations.

2 Related Work

2.1 Memory Representations for LLM Agents

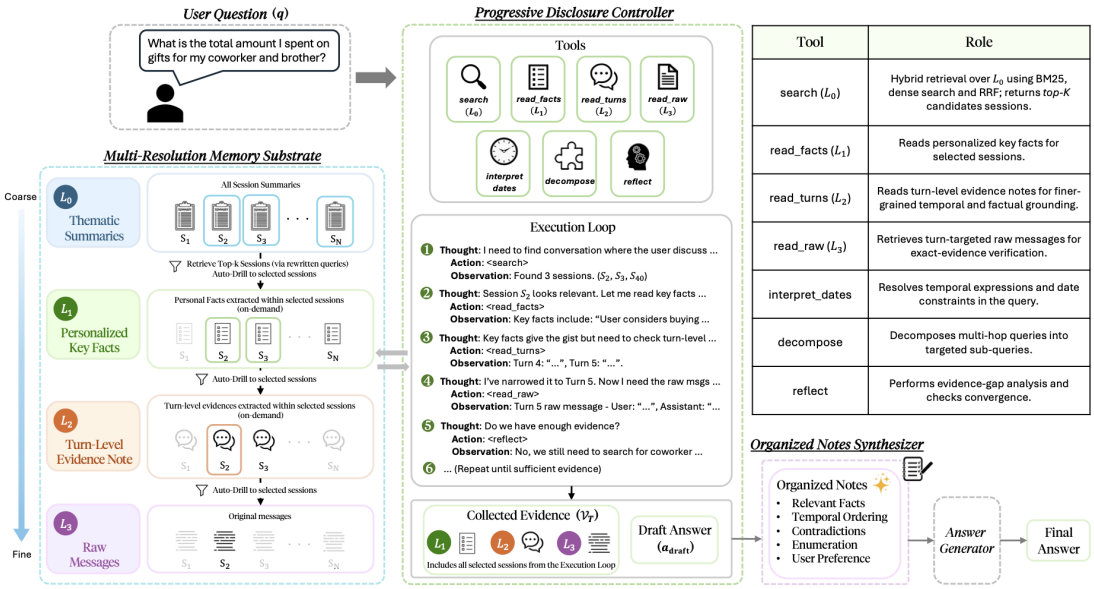
Personalized agents require persistent memory to maintain continuity across interactions (Zhang et al., 2026; Xu et al., 2026). Prior works frame this as a long-term memory problem: agents must retain user-specific facts and temporal context across sessions to avoid contextual amnesia (Chhikara et al., 2025; Fang et al., 2026; Liu et al., 2026). Memory systems differ mainly in how they represent past interactions. Profile- or fact-based methods store distilled user information (Zhong et al., 2024; Chhikara et al., 2025); episodic systems retrieve prior interaction traces (Packer et al., 2024; Xu et al., 2025); graph-based systems such as Zep organize temporal entity-relation memories (Rasmussen et al., 2025); and compression-based systems such as LightMem and SimpleMem reduce cost through staged consolidation or compact semantic memories (Fang et al., 2026; Liu et al., 2026). These approaches often require graph conversion, compression, or specialized memory objects, creating trade-offs between efficiency, fidelity, and complexity. APDMem instead stores conversations as transparent Markdown layers preserving the original conversation while providing compact abstractions for retrieval.

2.2 Progressive Disclosure

Memory systems organize information at different granularities—such as summaries, passages, chunks, or episodic traces—to improve retrieval over large contexts (Sarthi et al., 2024; Gao et al., 2023; Izacard and Grave, 2021; Khattab and Zaharia, 2020; Lewis et al., 2020; Asai et al., 2024; Trivedi et al., 2023). APDMem builds on this idea through *progressive disclosure* (Nielsen, 1993), exposing coarse memory views first and revealing finer evidence only when needed. To our knowledge, APDMem is the first system to make progressive disclosure an explicit architectural principle for agent memory, with an agent controlling disclosure depth through tool-mediated interactions.

2.3 Evidence-Grounded Generation

Complex queries involving temporal constraints, ambiguous references, user-specific details, multi-hop reasoning, or enumeration often require iterative retrieval and verification (Yao et al., 2023; Schick et al., 2023; Press et al., 2023; Trivedi et al., 2023). Prior work has shown the value of organizing or reasoning over retrieved evidence before answer generation (Yu et al., 2024; Gao et al., 2023; Asai et al., 2024). Inspired by Chain-of-Note (Yu et al., 2024), APDMem converts retrieved evidence into organized notes that consolidate relevant facts, temporal order, contradictions, enumeration evidence, and answer hints into a structured grounding context for the answer generator.



Tool	Role
search (L_0)	Hybrid retrieval over L_0 using BM25, dense search and RRF; returns *top-K* candidates sessions.
read_facts (L_1)	Reads personalized key facts for selected sessions.
read_turns (L_2)	Reads turn-level evidence notes for finer-grained temporal and factual grounding.
read_raw (L_3)	Retrieves turn-targeted raw messages for exact-evidence verification.
interpret_dates	Resolves temporal expressions and date constraints in the query.
decompose	Decomposes multi-hop queries into targeted sub-queries.
reflect	Performs evidence-gap analysis and checks convergence.

Figure 1: APDMem overview. The *Multi-Resolution Memory Substrate* stores conversations as layered views from thematic summaries (L_0) to raw messages (L_3). The *Progressive Disclosure Controller* uses tool calls to retrieve and drill down only as needed. The *Organized Notes Synthesizer* then consolidates collected evidence into structured notes for grounded answer generation. The upper-right panel summarizes tools available to the controller.

3 APDMem

As shown in Figure 1, APDMem consists of three components: (1) a *Multi-Resolution Memory Substrate* that stores each conversation across four layers of increasing abstraction (§3.1); (2) a *Progressive Disclosure Controller* that searches, drills into, and reflects over the memory hierarchy through tool-mediated interactions (§3.2); and (3) an *Organized Notes Synthesizer* that structures retrieved evidence for final answer generation (§3.3).

3.1 Multi-Resolution Memory Substrate

Given a conversation history, we partition it into N sessions $\mathcal{S} = S_1, \ldots, S_N$, where each session S_i is a contiguous conversation episode consisting of turns, and each turn is a user–assistant exchange. APDMem represents each session as a four-layer memory tuple, $S_i = (L_0, L_1, L_2, L_3)$. L_0 contains *thematic summaries* with resolved pronouns, normalized dates, keywords, and tags, and serves as the primary indexing layer. L_1 contains *personalized key facts*: session facts about user information, recommendations, plans, preferences, and constraints. L_2 contains *turn-level evidence notes*: turn-level factual bullets that preserve temporal order for fine-grained evidence reasoning. L_3 contains *raw messages*: the verbatim user–assistant messages retained for exact evidence recovery.[1]

Lazy materialization. Only L_0 and L_3 are materialized at ingestion time, whereas L_1 and L_2 are generated on demand when first requested by the progressive disclosure controller and then cached for reuse, reducing write-time cost.

3.2 Progressive Disclosure Controller

The *Progressive Disclosure Controller* implements memory access as a bounded, query-adaptive control process. Given a query q, the controller's objective is to gather sufficient evidence for answering q while avoiding unnecessary disclosure of finer memory. It maintains an evidence set $\mathcal{V}_t$ at step t. Each element in $\mathcal{V}$ is a retrieved view of a session S_i at abstraction level L_ℓ for $\ell \in \{0, 1, 2, 3\}$.

At each step, the controller selects a tool action $a_t \in \mathcal{A}$, where $\mathcal{A} = \{$`search`, `read_facts`, `read_turns`, `read_raw`, `interpret_dates`, `decompose`, `reflect`$\}$ is the action space. Memory-read actions correspond to different resolutions of the *Multi-Resolution Memory Substrate*: finer layers expose higher-fidelity evidence but incur greater token and latency cost, i.e., $c(L_0) < c(L_1) < c(L_2) < c(L_3)$ and $f(L_0) < f(L_1) < f(L_2) < f(L_3)$, where $c(\cdot)$ denotes access cost and $f(\cdot)$ denotes evidence

[1] Construction details, prompts, and real output examples for each layer are provided in Appendix I.

fidelity. The controller can therefore be viewed as optimizing a cost–fidelity trade-off:

$$\min_{a_{1:T}} \sum_{t=1}^{T} c(a_t) \quad \text{s.t.} \quad \text{sufficient}(\mathcal{V}_T, q) = 1, \quad (1)$$

where $\text{sufficient}(\mathcal{V}_T, q)$ indicates whether the collected evidence is adequate for answering the query. In practice, sufficiency is estimated through evidence reflection and bounded by the convergence criteria described in §3.2.2. Unlike flat retrieval systems, APDMem selects both the relevant memory region and the required evidence resolution.

3.2.1 Controller Workflow

A typical trajectory proceeds in three phases.

Query preparation. Queries with relative dates (e.g., "last summer") are normalized against the question timestamp via `interpret_dates`; multi-hop or compound queries are split into focused subqueries via `decompose`.

Progressive disclosure. `search` retrieves the top-K candidate sessions over L_0 (hybrid lexical + dense retrieval fused with RRF; Appendix G). An *auto-drill* step selects the most promising unexplored sessions and `read_facts` retrieves their L_1 key facts. If these facts are insufficient, the controller descends further: `read_turns` surfaces L_2 evidence in temporal order, and `read_raw` retrieves L_3 messages restricted to the most relevant turns within selected sessions. At each step, retrieved session information is stored in evidence set $\mathcal{V}$.

Reflection and drafting. APDMem uses reflection (Yao et al., 2023; Shinn et al., 2023) to avoid both premature answering and unbounded search. `reflect` summarizes $\mathcal{V}$, identifies missing evidence, and assesses whether additional retrieval is likely to help. If more evidence is needed, it proposes follow-up queries and the controller re-enters the progressive disclosure phase. Once $\mathcal{V}$ converges (§3.2.2), the controller produces a draft answer a_{draft} from the final evidence set $\mathcal{V}_T$, which is then passed to the Organized Notes Synthesizer.

3.2.2 Convergence Control

The controller loop terminates when (1) reflection finds no new evidence, $|\mathcal{V}_t| = |\mathcal{V}_{t-1}|$, or (2) step budget $T_{\max}$ is reached. $T_{\max}$ caps the maximum number of controller iterations and therefore bounds the additional tool calls the controller can make before it must stop escalating and return the best available answer.[2]

3.3 Organized Notes Synthesizer

The *Organized Notes Synthesizer* converts the query q, evidence set $\mathcal{V}$, and controller's draft answer a_{draft} into organized notes, a compact intermediate representation. Inspired by Chain-of-Note (Yu et al., 2024), this step uses the draft to verify, correct, or supplement the controller's initial reasoning. These notes organize relevant facts, temporal ordering, contradictions, enumeration evidence for counting queries, and an answer hint into a query-focused grounding context.

Answer generation. The answer generator receives the organized notes and a_{draft} to produce the final response. This synthesize-then-generate design preserves the controller's retrieval trajectory while allowing the final answer to correct errors or omissions surfaced during evidence synthesis.

4 Experiments

Experimental Setup. We evaluate on LongMemEval (Wu et al., 2025) (specifically the LongMemEval-S split): 500 questions across six categories (Table 1 column headers), each paired with a per-question session haystack. We compare against Full-context (no retrieval), Mem0 (Chhikara et al., 2025), LightMem (Fang et al., 2026), and SimpleMem (Liu et al., 2026). To ensure strict comparability, all baseline numbers are taken from Liu et al. (2026), and APDMem reuses the identical answer-generator prompt (Figure 10) and LLM-as-a-judge (GPT-4.1-mini, $temp$=0.0).[3] APDMem and each baseline are evaluated with GPT-4.1 and GPT-4.1-mini as backbones; for APDMem, a single backbone drives the *Progressive Disclosure Controller*, and the *Organized Notes Synthesizer*.[4]

4.1 Main Experiments

Table 1 presents the main results. APDMem reaches 87.8% with GPT-4.1 and 79.8% with GPT-4.1-mini, ahead of the best prior baseline (SimpleMem) by 3.8 and 2.9 points respectively.

[2] In all experiments, we set $T_{\max} = 20$. Appendix D reports the observed iteration distribution and discusses how to choose $T_{\max}$ for other applications.

[3] Following Fang et al. (2026); Liu et al. (2026), all numbers are with decoding temperature fixed at 0.0.

[4] Full implementation details and hyperparameters are provided in Appendix B, C, and G.

Backbone	Method	Temp.	Multi-S.	K-Upd.	SS-User	SS-Asst.	SS-Pref.	Avg.
GPT-4.1	Full-context*	51.9	39.1	70.5	65.7	96.4	16.7	56.7
	Mem0*	43.6	54.9	75.6	54.3	39.3	**83.3**	58.5
	RAPTOR†	38.4	51.9	68.0	95.7	92.9	60.0	62.0
	LightMem*	85.0	57.9	89.7	87.1	71.4	70.0	76.9
	SimpleMem*	86.5	**81.2**	80.8	**98.6**	76.8	80.0	84.0
	APDMem†	**90.2**	77.4	**91.0**	92.9	**100.0**	80.0	**87.8**
GPT-4.1-mini	Full-context*	27.1	30.1	41.0	47.1	32.1	60.0	39.6
	Mem0*	40.6	50.4	69.2	87.1	48.2	63.3	59.8
	RAPTOR†	38.4	55.6	69.2	90.0	**98.2**	66.7	63.4
	LightMem*	**85.7**	47.4	**92.3**	88.6	21.4	**76.7**	68.7
	SimpleMem*	83.5	60.9	79.5	85.7	75.0	**76.7**	76.9
	APDMem†	77.4	**66.2**	85.9	**91.4**	96.4	**76.7**	**79.8**

† denotes experiments run by us; * denotes prior results from Liu et al. (2026), reported under a unified protocol that we also follow for fair comparison.

Table 1: Performance (% accuracy) on LongMemEval-S (n=500). The highest value in each column is shown in **bold**; the second highest is underlined.

The gains are clearest on detail-oriented queries. For example, on SS-Asst, APDMem reaches 100.0% with GPT-4.1, surpassing SimpleMem (76.8%) and Full-context (96.4%) while inspecting only a small slice of the haystack (§4.3). This highlights the value of drilling to raw turns.

APDMem remains strong on multi-session synthesis with GPT-4.1, scoring 77.4%, though SimpleMem is higher at 81.2%. This reflects a breadth-depth trade-off: flat summary retrieval can expose more sessions per call, while APDMem investigates top-ranked candidates in greater detail. Notably, this is not a systematic weakness: with GPT-4.1-mini, APDMem achieves the best multi-session score, showing that targeted memory navigation can also benefit multi-session reasoning.

4.2 Inference Efficiency

We decompose cost into *offline* (one-time L_0 construction) and *online* (per-query controller cost, including on-demand L_1/L_2 generation). Following Fang et al. (2026), (a) all systems use GPT-4o-mini, (b) the shared answer generator is excluded from online cost, and (c) numbers are per-question averages on LongMemEval (n=500). Prior work reports only baseline construction cost, so online cost is available only for APDMem; baseline online cost is marked unavailable (-) in Table 2.[5]

Table 2 compares APDMem with two classes of prior memory systems. Update-heavy methods such as Mem0 and A-MEM rely on iterative memory maintenance or consolidation, incurring million-token and multi-thousand-second construction costs; APDMem uses roughly one-tenth of their offline token budget and a small fraction of their latency. By contrast, LightMem reduces offline tokens through query-agnostic pre-compression, which can discard fine-grained details. APDMem offers a practical efficiency–accuracy trade-off: it avoids the high overhead of update-heavy systems while reducing reliance on aggressive compression, with minimal latency.

[5]See Appendix H for a per-backbone breakdown.

		Offline (Construction)		Online (Controller)	
Method	ACC	Tok. (k)	Lat. (s)	Tok. (k)	Lat. (s)
Mem0	53.6	1,152.6	4,248.5	–	
A-MEM	62.6	1,605.8	5,132.1	–	
LightMem	68.6	28.3	283.8	–	
APDMem	74.4	145.4	101.5	124.3	37.7

Table 2: Efficiency on LongMemEval-S (GPT-4o-mini). *Offline*: memory construction; *Online*: controller cost (cold start: no cached state for L_1 & L_2). Baselines from Fang et al. (2026) (no online numbers available).

4.3 Effectiveness of Progressive Disclosure

We analyze whether progressive disclosure achieves its design goal: locating relevant evidence with minimal memory access. Unless stated otherwise, statistics use the GPT-4.1 configuration; the error decomposition additionally contrasts GPT-4.1-mini with GPT-4.1 to study how backbone strength affects retrieval versus generation failures.

Retrieval selectivity. Figure 2(a) shows that the controller drills into only **8%** of the haystack on average (3.7 out of 47.7 sessions per question). Single-session categories are the most selective (SS-Asst.: 2%, SS-User: 6%), while multi-session questions require broader access (12%). Despite this selectivity, APDMem achieves 87.8% overall accuracy, confirming that the *Progressive Disclosure Controller* enables precise evidence identification without exhaustive search.

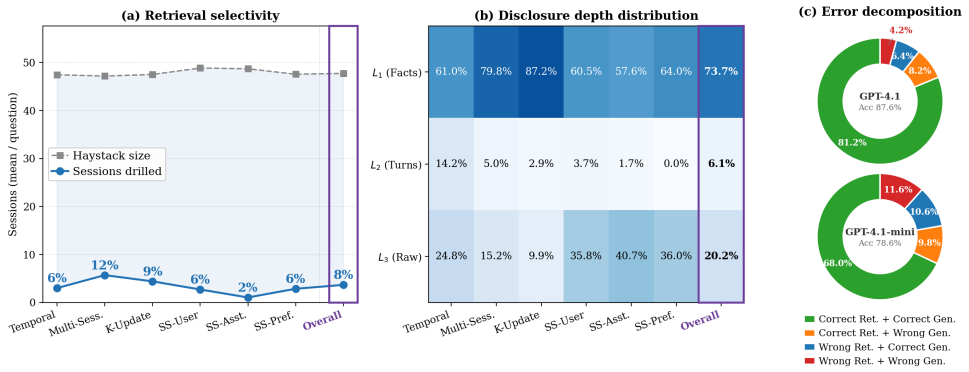


Figure 2: Progressive disclosure behavior (GPT-4.1). (a) Retrieval selectivity: fraction of haystack sessions drilled per category. (b) Disclosure depth distribution across memory levels. (c) Error decomposition (retrieval vs. generation) for GPT-4.1-mini and GPT-4.1.

Controller	Generator	Avg. Acc. (%)
GPT-4.1-mini	GPT-4.1-mini	79.8
GPT-4.1	GPT-4.1-mini	86.8
GPT-5.2	GPT-4.1-mini	87.8

Table 3: Effect of controller backbone strength with a fixed generator on LongMemEval-S.

Disclosure depth distribution. Figure 2(b) shows that the controller usually stops shallow: among 1,838 drills, 73.7% terminate at L_1, 20.2% descend to L_3, and 6.1% stop at L_2. Depth varies by question type: single-session queries use L_3 most often (SS-Asst. 40.7%, SS-User 35.8%) because they often require exact phrasing, whereas multi-session and knowledge-update queries remain mostly shallow (L_3 rates of 15.2% and 9.9%). This demonstrates that the controller adapts disclosure depth to evidence demand rather than using a fixed retrieval granularity.

Error decomposition. Using gold session annotations, we classify predictions by *Correct Retrieval* and *Correct Generation*. Figure 2(c) shows that scaling the backbone primarily reduces retrieval errors (22.2% → 10.6%), while generation errors remain near 8–10%, making evidence discovery the main bottleneck. This is confirmed by Table 3: with the generator fixed to GPT-4.1-mini, upgrading only the controller raises accuracy from 79.8% to 86.8%, and a GPT-5.2 controller reaches 87.8%. Thus, APDMem enables heterogeneous deployment: when model budget is limited, capacity can be allocated to the controller, where the dominant bottleneck lies, while a lightweight model serves as the generator.

Configuration	Avg. Acc. (%)	Diff
APDMem	87.8	—
Vanilla RAG (L_3-only)	77.0	↓ 12.3%
Controller ablation		
w/o *Controller*†	83.2	↓ 5.2%
Memory hierarchy ablations		
w/o `read_facts` (L_1)	84.6	↓ 3.6%
w/o `read_turns` (L_2)	87.4	↓ 0.5%
w/o `read_raw` (L_3)	82.6	↓ 5.9%
Reasoning tool ablations		
w/o `reflect`	84.6	↓ 3.6%
w/o `decompose`	86.0	↓ 2.1%
w/o `interpret_dates`	87.6	↓ 0.2%

†RAG retrieves top-K L_0 sessions; same sessions are populated in L_1–L_3

Table 4: Ablation on memory hierarchy levels and reasoning tools (GPT-4.1). The "Diff" column is the percentage drop with respect to APDMem.

4.4 Ablation Studies

We ablate three aspects of APDMem on GPT-4.1 in Table 4: the controller, memory levels, and reasoning tools. To further disentangle the effect of hierarchical memory from agentic control, we additionally include a vanilla RAG (L_3-only) baseline, which removes both the controller and the hierarchy and retrieves directly from raw messages. A complementary ablation of the *Organized Notes Synthesizer* is reported in Appendix J.

Results. Three trends emerge, with drops reported relative to APDMem. (i) *Controller and hierarchy*: replacing the controller with a non-agentic RAG baseline while preserving the L_0–L_3 hierarchy reduces accuracy from 87.8% to 83.2%, corresponding to a 5.2% relative drop. This shows that query-adaptive drilling and reflection add value over static retrieval. The vanilla RAG (L_3-only) baseline further removes the hierarchy and retrieves

Case 1: Multi-hop across sessions
Q: "How old was I when Alex was born?" (gold: 11)

Find first fact. `search`(L_0, "Alex birth year") → S9; `read_facts`(L_1, S9) → *"Alex is 21"*

Identify missing evidence. `reflect`(): need user's age.

Find second fact. `search`(L_0, "my age in 2023") → S20; `read_facts`(L_1, S20) → *"User turned 32 in 2023-04"*

Draft and verify. Draft: $32 - 21 = 11$.
`reflect`(): verify with broader search.
`search`(L_0, "family timeline") → S0, S16, S38, S33;
`read_facts`(L_1): no new evidence.

Converge. `reflect`(): sufficient → Final answer: 11 ✓

Only L_1 evidence read; 6 sessions drilled.

Case 2: Verbatim recall within a session
Q: "Orlando dessert shop with giant milkshakes?"
(gold: Sugar Factory at Icon Park)

Locate candidate session. `search`(L_0, "Orlando giant milkshakes") → S10;
`read_facts`(L_1, S10) → topic ✓, no shop name.

Drill for finer evidence. `read_turns`(L_2, S10) → relevant turn, still no exact name.

Recover verbatim detail. `read_raw`(L_3, S10, "milkshakes") →
"Sugar Factory at Icon Park"

Converge. Final answer: Sugar Factory at Icon Park ✓

Descends $L_1 \rightarrow L_2 \rightarrow L_3$ in 1 session.

Figure 3: Two representative APDMem traces. Case 1 answers a cross-session multi-hop question using only L_1 facts, with `reflect` driving evidence-gathering and convergence. Case 2 answers a within-session verbatim question by descending the hierarchy from L_1 to L_3. The two cases highlight the breadth-vs.-depth behavior enabled by the controller.

only from raw messages (L_3), reducing accuracy to 77.0%, corresponding to a larger 12.3% relative drop. This comparison separates the contribution of the two components: the controller improves over hierarchical RAG, while the hierarchy itself provides a substantial gain over raw-message retrieval. (ii) *Memory levels*: key facts (L_1, ↓3.6%) and raw messages (L_3, ↓5.9%) cause the largest accuracy drops, consistent with APDMem's broad-shallow and narrow-deep retrieval patterns. Turn-level evidence notes (L_2, ↓0.5%) have a smaller aggregate effect, serving primarily as an intermediate evidence layer. (iii) *Reasoning tools*: reflection contributes most (↓3.6%), followed by decomposition (↓2.1%); date interpretation has little effect (↓0.2%), likely because the backbone model already handles many temporal cues.

4.5 Case Studies

Figure 3 shows two representative LongMemEval traces illustrating how APDMem performs progressive disclosure in different scenarios.

Case 1. The first retrieved session provides Alex's age but not the user's age, so `reflect` identifies the missing evidence and triggers a second search. After deriving the draft answer, a broader L_1 check over four additional sessions finds no new evidence, allowing the controller to converge without descending to L_2 or L_3.

Case 2. The controller first localizes the relevant Orlando session, but L_1 only confirms the topic and does not preserve the specific shop name. APDMem therefore drills progressively: L_2 narrows the evidence region, and L_3 recovers the verbatim detail needed for the final answer.

These traces illustrate APDMem's adaptive behavior: the controller expands across sessions when evidence is scattered, but descends within a session when the answer depends on fine-grained details.

5 Conclusion

We present **APDMem**, a lightweight hierarchical memory architecture that frames long-term memory access as a query-adaptive disclosure problem. A controller first reads coarse abstractions and drills into finer-grained evidence only when needed, while a notes synthesizer organizes the surfaced evidence for grounded answer generation. On LongMemEval, APDMem achieves a substantial improvement over prior work, reaching 87.8% accuracy with GPT-4.1. Our key findings are: (i) disclosure depth adapts to question type, with most questions answered from the high-level key-fact layer, while certain detail-oriented question categories benefit from raw-message access; (ii) controller ablations show that agent-controlled progressive disclosure is a significant contributor to performance; and (iii) APDMem offers a practical efficiency–accuracy trade-off, balancing token cost and latency while maintaining strong answer accuracy.

Limitations

APDMem's retrieval recall depends on the quality of the L_0 thematic summaries: if a summary omits a relevant detail, the corresponding turn may receive lower BM25 and dense-retrieval scores. The keyword-field boost (Appendix G) and the

controller's `reflect` tool provide partial recovery mechanisms, but initial ranking remains sensitive to summary fidelity. To this end, developing a systematic summary-quality evaluation and monitoring for L_0 is a natural and important direction for future work. As one potential mitigation, summary quality can be assessed using an LLM-as-a-judge style evaluation (e.g., checking faithfulness and coverage against the underlying raw messages).

Lazy materialization of L_1 and L_2 may introduce cold-start latency when a session is accessed for the first time. To mitigate first-access latency, L_1 and L_2 can be proactively materialized (e.g., offline or in the background). That said, our design intentionally follows progressive disclosure and lazy materialization: precomputing everything can be unnecessarily costly when many sessions (and fine-grained layers) are never queried. In production, this becomes an explicit cost–latency tradeoff: deployments that prioritize low tail-latency can pre-warm likely-to-be-accessed sessions, while cost-sensitive settings can keep on-demand generation to minimize wasted computation.

Finally, APDMem does not explicitly model cross-session relationships in the way graph-based memory structures can. Exploring lightweight mechanisms for cross-session linking is a promising direction for future work.

Disclaimer

This paper was prepared for informational purposes in part by the Machine Learning Center of Excellence group of JPMorgan Chase & Co. and its affiliates ("JP Morgan") and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

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*.

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*.

Gordon V Cormack, Charles LA Clarke, and Stefan Buettcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 758–759.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. 2026. Lightmem: Lightweight and efficient memory-augmented generation. In *The Fourteenth International Conference on Learning Representations*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online. Association for Computational Linguistics.

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 39–48, New York, NY, USA. Association for Computing Machinery.

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. Curran Associates, Inc.

Dongshuo Liu, Zhijing Wu, Dandan Song, and Heyan Huang. 2025. A persona-aware LLM-enhanced framework for multi-session personalized dialogue generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 103–123, Vienna, Austria. Association for Computational Linguistics.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. 2026. Simplemem: Efficient lifelong memory for llm agents. *Preprint*, arXiv:2601.02553.

Jakob Nielsen. 1993. *Usability Engineering*. Morgan Kaufmann, San Francisco, CA.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Memgpt: Towards llms as operating systems. *Preprint*, arXiv:2310.08560.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, Singapore. Association for Computational Linguistics.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*.

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

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551. Curran Associates, Inc.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2025. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth 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, Main Conference, pages 17577–17604. Curran Associates, Inc.

Yue Xu, Qian Chen, Zizhan Ma, Dongrui Liu, Wenxuan Wang, Xiting Wang, Li Xiong, and Wenjie Wang. 2026. Toward personalized llm-powered agents: Foundations, evaluation, and future directions. *arXiv preprint arXiv:2602.22680*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Wenhao Yu, Hongming Zhang, Xiaoman Pan, Peixin Cao, Kaixin Ma, Jian Li, Hongwei Wang, and Dong Yu. 2024. Chain-of-note: Enhancing robustness in retrieval-augmented language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14672–14685, Miami, Florida, USA. Association for Computational Linguistics.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. 2026. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 21457–21483, San Diego, California, United States. Association for Computational Linguistics.

Weizhi Zhang, Xinyang Zhang, Chenwei Zhang, Liangwei Yang, Jingbo Shang, Zhepei Wei, Henry Peng Zou, Zijie Huang, Zhengyang Wang, Yifan Gao, Xiaoman Pan, Lian Xiong, Jingguo Liu, Philip S. Yu, and Xian Li. 2026. PersonaAgent: Bridging memory and action for personalized LLM agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 26421–26439, San Diego, California, United States. Association for Computational Linguistics.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 19724–19731.

A Progressive Disclosure Controller Algorithm

Algorithm 1 summarizes the iterative protocol of progressive disclosure controller.

B System Prompts

This appendix provides all system prompts used in APDMem. Figures 4–6 show the memory construction prompts for L_0 - L_2; Figures 7–10 show the controller, evidence reflection, organized notes synthesis, and answer generation prompts.

Algorithm 1 Progressive Disclosure Controller

Require: Query q, timestamp d, memory $\mathcal{S}$, max steps $T_{\max}$
$\mathcal{V}_0 \leftarrow \emptyset$; nudges $\leftarrow 0$
for $t = 1$ **to** $T_{\max}$ **do**
  response $\leftarrow$ LLM(q,d,$\mathcal{V}_{t-1}$,$\mathcal{A}$)
  if response contains tool calls **then**
    $\mathcal{V}_t \leftarrow \mathcal{V}_{t-1}$
    for all tool call τ in response **do**
      $\mathcal{V}_t \leftarrow \mathcal{V}_t \cup \textsc{Execute}(\tau)$
    end for
  else if $|\mathcal{V}_{t-1}|$ grew since last reflection **and** nudges < 2 **then**
    Inject nudge message; nudges $\leftarrow$ nudges $+1$
    $\mathcal{V}_t \leftarrow \mathcal{V}_{t-1}$
  else
    $a_{\text{draft}} \leftarrow$ response
    return $a_{\text{draft}}, \mathcal{V}_{t-1}$
  end if
end for
return $a_{\text{draft}}, \mathcal{V}_{T_{\max}}$

```
You are a memory-indexing assistant. For each
user<->assistant turn in the conversation,
produce:
1. A 1-sentence summary (max 30 words).
2. keywords (5-8): specific names, items, dates,
   locations, actions mentioned.
3. tags (3-5): broader category labels.

Rules:
- NO pronouns -- use actual names/entities.
- NO relative dates -- convert to absolute dates
  using the [Session date] anchor.
- Each summary must be self-contained.

Session date: {session_date}

[conversation window]

Return JSON:
{"turns": [{"turn_number": 1,
  "summary": "...",
  "keywords": ["kw1", ...],
  "tags": ["tag1", ...]}, ...]}
```

Figure 4: Prompt for L_0 turn summary and keyword generation (§3.1).

Memory construction prompts. All three construction prompts (Figures 4–6) enforce semantic disambiguation: pronouns are resolved to named entities and relative dates are converted to absolute dates using the session timestamp as anchor.

Retrieval and reasoning prompts. The controller (Figure 7) operates the progressive disclosure loop, selecting tools based on question type and evidence sufficiency. The reflection prompt (Figure 8) performs structured gap analysis and generates follow-up queries when evidence is insufficient.

Evidence organization and answer generation. The organized notes synthesizer prompt (Figure 9) instructs the model to structure retrieved evidence by relevance, temporal order, contradictions, and enumeration without producing a final answer. The answer generator (Figure 10) then independently verifies the evidence and produces the final response; this prompt is shared with SimpleMem for fair comparison.

```
You are a memory-management assistant. Given a
conversation between a user and an assistant,
extract a bullet list of key facts, events,
preferences, and knowledge mentioned.

Semantic Disambiguation Rules (CRITICAL):
1. PROHIBIT all pronouns (he, she, it, they).
   Always use the actual person/item name.
2. PROHIBIT relative time expressions.
   Convert ALL time references to absolute
   dates using the [Session date] anchor.
3. Each bullet must be a complete,
   self-contained, lossless restatement
   understandable without the conversation.
4. Include specific: dates, person names, item
   names, locations, quantities, preferences.
5. Do NOT omit any factual information.

Return ONLY the bullet list (use "- " prefix).
Aim for 5-25 bullets depending on richness.
```

Figure 5: Prompt for L_1 key fact extraction (§3.1).

```
You are a memory-management assistant. Given a
conversation between a user and an assistant,
produce per-turn atomic memory entries -- multiple
self-contained fact bullets for each user<->
assistant exchange (turn).

Semantic Disambiguation Rules (CRITICAL):
1. PROHIBIT all pronouns (he, she, it, they).
   Always use the actual person/item name.
2. PROHIBIT relative time expressions.
   Convert ALL time references to absolute
   dates using the [Session date] anchor.
3. Each bullet must be a complete, independent,
   lossless restatement -- a single atomic fact
   fully understandable without other context.
4. Include specific: dates, person names, item
   names, locations, quantities, preferences.
5. Do NOT merge multiple facts into one bullet.

Output format:

Turn 1:
- <atomic fact 1>
- <atomic fact 2>

Turn 2:
- <atomic fact 1>
- <atomic fact 2>
```

Figure 6: Prompt for L_2 turn-level evidence generation (§3.1).

C Hyperparameter Configuration

Table 5 lists all hyperparameters used in the APDMem experiments. Memory construction parameters (L_0 window size and L_1 bullet bounds) are fixed across all backbone models. Retrieval parameters were selected based on preliminary experiments on a held-out development set of 20 questions and kept constant for all reported results.

```
You are a personal-memory retrieval agent.
The user previously had many conversations with
an AI assistant. Those conversations have been
stored in a multi-level memory system.

Memory levels (progressive disclosure):
 Level 0 -- session summaries (per-turn
   one-liners + keywords)
 Level 1 -- key facts (bullet list extracted
   from a session)
 Level 2 -- turn summaries (self-contained,
   no pronouns, absolute dates)
 Level 3 -- raw messages (actual user<->
   assistant text)

STRATEGY GUIDELINES:
1. If the question contains relative time
   references, call interpret_dates FIRST.
2. If the question is complex, call
   decompose.
3. Call search with 3-5 diverse queries.
4. READ key facts for ALL possibly-relevant
   sessions. For counting questions, cast a
   WIDE net.
5. For counting/aggregation questions: MUST call
   reflect to check for gaps.
6. Use temporal reasoning: recent sessions'
   items likely qualify.
7. Deduplicate items across sessions.

CRITICAL: For counting questions, NEVER give
a final answer without calling
reflect at least once.
```

Figure 7: System prompt for the Progressive Disclosure Controller (§3.2).

```
You are a reflection assistant for a memory
retrieval system that uses BM25 keyword matching
with English stemming and embedding similarity.

The user asked (on {question_date}): {question}

Sessions searched: {all_sessions}
Sessions drilled (key facts read): {drilled}
Previous queries tried: {prev_queries}

Key facts gathered:
{gathered}

Your task:
1. ASSESS: Are facts sufficient for a COMPLETE
   answer?
2. CONVERGENCE CHECK: If same sessions keep
   appearing and no new ones found, mark
   sufficient.
3. ENTITY EXPANSION: Extract entities from
   facts as potential new search terms.
4. If NOT sufficient, generate 1-5 NEW search
   queries (3-8 words, different vocabulary
   from all previous queries).

Return JSON:
{"is_sufficient": true/false,
 "items_found_so_far": [...],
 "potential_gaps": "...",
 "new_queries": [...]}
```

Figure 8: Prompt for the `reflect` tool (§3.2).

D Setting the Controller Step Budget

The controller step budget $T_{\max}$ is an explicit inference-time budget parameter. It caps the maximum number of convergence-control iterations and therefore bounds the maximum number of additional tool/LLM calls the controller can make before it must stop escalating and return the best available answer.

Experimental setting. In all experiments, we set $T_{\max} = 20$. This is intentionally a generous cap: when the controller converges earlier, it terminates before reaching the budget, while $T_{\max}$ prevents rare difficult queries from incurring unbounded cost or latency. Table 6 reports the observed iteration distribution under this setting.

With $T_{\max} = 20$, nearly all queries terminate before reaching the cap. Thus, in our experiments, the budget does not materially affect answer quality for most examples, but it still provides an explicit worst-case compute bound.

Choosing $T_{\max}$ in other applications. $T_{\max}$ controls a quality–cost trade-off: larger values allow the controller to continue searching and reflecting on difficult queries, whereas smaller values reduce worst-case latency and tool/LLM usage. In practice, we recommend setting $T_{\max}$ as follows:

1. **Profile demand.** On a representative sample, measure the empirical iteration distribution and set $T_{\max}$ near a high percentile, such as p95–p99, so truncation is rare.
2. **Apply operational bounds.** Ensure the resulting maximum stays within deployment constraints, including latency and cost budgets.
3. **Choose the tightest sufficient bound.** Use the smallest $T_{\max}$ that keeps truncation rare while satisfying the operational constraints.

E Statistical Significance Analysis

To further validate the robustness of the main results, we conduct a controlled multi-run evaluation comparing APDMem with SimpleMem, the strongest prior baseline. Both methods are evaluated on the full LongMemEval-S benchmark, which contains 500 questions, using GPT-4.1 as the backbone model. We run each method for 10 independent runs under the same evaluation protocol and report the mean and standard deviation of accuracy across runs.

APDMem achieves an average accuracy of 87.5%, compared with 84.1% for SimpleMem, yielding an average improvement of +3.4 percentage points. To test whether this improvement is statistically significant, we report both a parametric significance test and a non-parametric confidence interval.

```
Read the question and all the retrieved evidence below. Produce organized notes that will help
an answer generator answer the question correctly.

## Question (asked on {question_date})
{question}

## Retrieved Evidence
{evidence_text}

## Instructions
Produce your notes following this structure:

1. **RELEVANT FACTS**: List every fact from the evidence that is directly relevant to answering
   the question. For each fact, note which session it comes from and the session date.
   Order by relevance (most relevant first).

2. **TEMPORAL ORDERING** (if dates matter): Arrange relevant facts chronologically. Clearly mark
   which is the MOST RECENT. If the question asks about the current/latest state, highlight it.

3. **CONTRADICTIONS / UPDATES**: If multiple sessions contain conflicting information about
   the same topic, note ALL versions with their dates. The LATEST date = current truth.

4. **ENUMERATION** (if "how many" or "list all"): Enumerate every qualifying item found.
   Deduplicate items appearing in multiple sessions. Flag uncertain items.

5. **ANSWER HINT**: Based on your organization of the evidence, what answer does the
   evidence point to? (This is a hint, not a final answer.)

Rules: Include EVERY relevant detail -- do not summarize away specifics (names, numbers,
dates, titles, locations). Be exhaustive for counting questions.
```

Figure 9: Prompt for Organized Notes Synthesizer (§3.3). The system message instructs the model to organize evidence without answering the question itself.

```
System: You are a professional Q&A assistant.
Extract concise answers from context.
You must output valid JSON format.

User:
Answer the user's question based on the
provided context.

User Question (asked on {question_date}):
{query}

Relevant Context:
{context_str}

Requirements:
1. Think through the reasoning process.
2. Provide a CONCISE but COMPLETE answer --
   include specific names, titles, key details.
3. Answer must be based ONLY on context.
4. Independently scan ALL context entries.
   Do NOT simply trust the Agent's preliminary
   answer -- verify it yourself.
5. Return JSON format.

Output Format:
{"reasoning": "Brief thought process",
 "answer": "Concise answer in a short phrase"}
```

Figure 10: Answer generator prompt (§3.3). Identical to SimpleMem's prompt for fair comparison; only the input context differs.

Module	**Parameter**	**Value**
L_0	Turn window size W	4 turns
L_0	Window overlap	1 turn
L_0	Max chars/window	6000
L_1	Max input chars	12000
L_1	Bullet count	5–25
L_1	Generation	On-demand
L_1	Workers	8
Embedding	Model	Nomic-v1.5
Embedding	Dimension	768
Retrieval	RRF constant k	60
Retrieval	Cosine weight $w_{\cos}$	0.5
Retrieval	Top-K sessions	20
Retrieval	Keyword boost	0.1
Agent	Max ReAct steps	20
Agent	Auto-drill/search	3
Agent	Max drill sessions	10
Agent	Max raw turns	20
Agent	Max reflection rounds	2
LLM	Agent temperature	0.0

Table 5: Hyperparameter configuration for APDMem.

As the primary parametric test, we use Welch's t-test, which compares the run-level means while allowing the two methods to have unequal variances. As a complementary robustness check, we also compute a bias-corrected and accelerated bootstrap confidence interval, which makes fewer distributional assumptions. The results are shown in Table 8.

Both confidence intervals are well above zero and provide consistent estimates of the APDMem–SimpleMem performance gap. These results indicate that, when evaluated using GPT-4.1, the observed improvement is statistically significant and unlikely to be explained by run-to-run variation alone.

Model	Median	p95	p99	Max
GPT-4.1	5	12	17	20
GPT-4.1-mini	3	12	16	19

Table 6: Observed controller iteration statistics with $T_{\max} = 20$. The cap is reached only in rare cases, so it primarily acts as a worst-case cost and latency bound.

Method	Accuracy (%)
APDMem	87.5 $\pm$1.0
SimpleMem	84.1 $\pm$0.7

Table 7: Multi-run accuracy on the full LongMemEval-S benchmark using GPT-4.1. We report mean and standard deviation over 10 independent runs. APDMem improves over SimpleMem by +3.4 percentage points on average.

F Human Validation of the LLM-as-a-Judge

All accuracy results in our main experiments use the LongMemEval LLM-as-a-judge protocol. Following prior work, we use the official judge prompt with GPT-4.1-mini at temperature 0.0, applied uniformly to APDMem and all baselines. Since our reported gains depend on this evaluation protocol, we conducted an additional human validation study to verify that the judge is reliable in our setting.

Human verification. We manually validated a stratified subset of 120 question–prediction pairs, with 20 examples from each of the six LongMemEval-S categories. One author, blind to the judge decision, labeled each prediction as CORRECT or WRONG according to whether it matched the gold answer, using the same criterion as the judge. To stress-test reliability, we over-sampled examples that the judge labeled WRONG, enriching the subset with borderline cases where disagreements are more likely.

Agreement. Human and judge labels agreed on 119 of 120 examples, corresponding to 99.2% agreement and Cohen's $\kappa = 0.98$, indicating near-perfect agreement. Agreement was consistently high across all categories, as shown in Table 9.

The single disagreement was a case where the human annotator marked the prediction as correct while the judge marked it as wrong, suggesting that the judge is not biased in favor of APDMem. Overall, this validation provides direct evidence that the LongMemEval judge is reliable in our setting and mitigates the concern that the reported gains are driven by an unvalidated LLM-as-a-judge protocol.

G Hybrid Retriever

The Hybrid Retriever retrieves candidate sessions using two complementary indexes over all L_0 thematic-summary units $\{c_t\}$.

Lexical index. A BM25 inverted index with stemming indexes each c_t and its keyword set $\mathcal{K}_t$ as separate fields, capturing exact matches for entities, proper nouns, actions, and dates.

Dense index. Each c_t is embedded with a sentence encoder f_{enc} using task-specific prefixes. Embeddings are L2-normalized, reducing dense retrieval to dot-product search.

Reciprocal Rank Fusion. Given a query q, the lexical and dense indexes each produce a ranking over turns. The Hybrid Retriever combines them using Reciprocal Rank Fusion (RRF) (Cormack et al., 2009):

$$\text{rrf}(q, c_t) = \sum_{j \in \{\text{lex,dense}\}} \frac{w_j}{k + r_j(c_t)}, \tag{2}$$

where $r_j(c_t)$ is the 1-based rank from index j, k is a smoothing constant, and w_j are equal weights.

Session scoring. Each session is scored by its best-matching turn:

$$s(q, S_i) = \max_{t \in S_i} \text{rrf}(q, c_t). \tag{3}$$

To preserve lexical precision for entity-heavy queries, the Hybrid Retriever adds a keyword-field boost for exact matches:

$$s'(q, S_i) = s(q, S_i) + \alpha \cdot s_{\max} \cdot n_i, \tag{4}$$

where $s_{\max}$ is the top hybrid score, n_i is a discretized match count, and α is a small constant.

Multi-query fusion. For decomposed questions, the controller may issue multiple queries $Q = \{q_1, \ldots, q_m\}$. The Hybrid Retriever scores each session by its best match across the query set:

$$s_{\text{final}}(S_i) = \max_{q_j \in Q} s'(q_j, S_i). \tag{5}$$

The top-K sessions are returned with their thematic summaries.

Test	Type	95% CI of Δ	Statistic / Details
Welch's t-test	Parametric	$[+2.55, +4.25]$	$t = 8.44, p = 2.6 \times 10^{-7}$
BCa bootstrap	Non-parametric	$[+2.70, +4.20]$	$B = 10{,}000$

Table 8: Statistical significance analysis of the APDMem–SimpleMem accuracy gap on LongMemEval-S using GPT-4.1. Here, Δ denotes the accuracy difference between APDMem and SimpleMem in percentage points. Both confidence intervals are strictly above zero, indicating that the improvement is statistically significant.

Category	n	Agree.	κ
knowledge-update	20	0.950	0.886
multi-session	20	1.000	1.000
single-session-assistant	20	1.000	1.000
single-session-preference	20	1.000	1.000
single-session-user	20	1.000	1.000
temporal-reasoning	20	1.000	1.000
Overall	120	0.992	0.980

Table 9: Human validation of the LongMemEval judge on 120 stratified question–prediction pairs.

	Offline (Construction)			Online (Controller)			
Model	**In**	**Out**	**Tot.**	**In**	**Out**	**Tot.**	**Lat.**
GPT-4o-mini	130.1	15.2	145.4	121.5	2.8	124.3	37.7s
GPT-4.1-mini	130.1	20.6	150.7	86.9	2.4	89.3	57.7s
GPT-4.1	130.1	21.2	151.3	125.3	3.2	128.5	45.9s
GPT-5.2	130.1	23.7	153.8	84.8	1.7	86.5	42.7s

Table 10: Per-model efficiency breakdown (per-question averages, LongMemEval n=500). Input/output token counts and total are in thousands (k). Offline cost is identical across models except for output tokens; online cost reflects the agentic controller only.

H Per-Model Efficiency Breakdown

Table 10 reports the per-model token usage and latency for both offline memory construction and online retrieval. All numbers are per-question averages on LongMemEval (n=500).

Offline cost. All models share the same input cost (130.1k tokens) because L_0 generation processes identical conversation windows. Output tokens increase slightly with model capability (15.2k for GPT-4o-mini vs. 23.7k for GPT-5.2), reflecting richer summaries and keywords, but the total remains within a narrow 145–154k range.

Online cost. Stronger models exhibit more efficient retrieval: GPT-4.1-mini and GPT-5.2 consume only 86–89k tokens compared to 121–129k for GPT-4o-mini and GPT-4.1. This suggests that more capable models require fewer ReAct steps to converge, they generate better initial search queries and identify evidence gaps earlier, reducing iterative exploration. GPT-5.2 achieves the lowest token cost (86.5k) *and* competitive latency (42.7s), indicating that its retrieval efficiency translates to both cost and speed improvements. GPT-4.1-mini shows the highest latency (57.7s) despite low token usage, likely due to slower per-request API response times.

I Multi-Resolution Memory Substrate: Construction and Examples

This section describes the construction process for each layer of the Multi-Resolution Memory Substrate (§3.1) and provides real output examples from the LongMemEval dataset. Prompts for L_0, L_1, and L_2 are shown in Figures 4, 5, and 6, respectively. All examples below are from the same user (`0a995998`, 44 sessions).

I.1 L_0: Thematic Summaries

Construction. Each session is split into user–assistant turns. A sliding window of W=4 turns (overlap 1) is passed to the LLM, which produces a one-sentence summary (max 30 words), 5–8 keywords, and 3–5 tags per turn. Pronouns are resolved and relative dates are converted to absolute dates using the session timestamp as anchor. Summaries are stored in Markdown and embedded with a sentence encoder for dense retrieval. L_0 is materialized at ingestion time. Figure 11 shows a real L_0 output.

I.2 L_1: Personalized Key Facts

Construction. When the progressive disclosure controller calls `read_facts` on a session, the LLM extracts a bullet list of 5–25 self-contained key facts about user information, preferences, plans, and constraints. Each bullet resolves all pronouns and relative dates. Generated facts are cached for subsequent accesses. L_1 is generated on demand (lazy materialization). Figure 12 shows a real L_1 output.

```
## Session 0 (date=2023/02/15, id=f10be626)

Turn 1:
- Summary: 2023-02-15 conversation covered
  solo-trip destination ideas for user with
  $2000 budget over 5-7 days.
- Keywords: 2023-02-15, solo trip planning,
  $2000 budget, 5-7 days, Airbnb, Southeast
  Asia, Portugal, Costa Rica
- Tags: travel planning, budgeting,
  accommodations, solo travel

Turn 2:
- Summary: Assistant recommended Lisbon and
  Porto in Portugal with estimated daily
  costs of $80-120.
- Keywords: Lisbon, Porto, Portugal, daily
  budget $80-120, hostels, local food
- Tags: destination recommendation, Europe,
  budget travel
```

Figure 11: L_0 output example: per-turn thematic summaries with keywords and tags.

```
## Session 2 (date=2023/02/15, id=750b437b_3)

- User was looking for recipe ideas that
  incorporate seasonal ingredients.
- User made a batch of roasted root
  vegetables before 2023-02-15.
- User was interested in citrus fruit
  recipes, including oranges and lemons.
- User stated that oranges and lemons are
  in season from 2022-12 to 2023-05.
- User was particularly interested in
  trying Citrus and Avocado Salad.
- User was particularly interested in
  trying Lemon and Herb Roasted Chicken.
- User planned to try Citrus and Avocado
  Salad for a lunch break at work.
- User usually took a 10-minute walk
  outside during a work break.
- User spotted daffodils blooming in a
  nearby park before 2023-02-15.
- User had been experimenting with spring
  desserts including lemon bars.
```

Figure 12: L_1 output example: self-contained key facts extracted from a single session.

I.3 L_2: Turn-Level Evidence Notes

Construction. When the controller calls `read_turns` on a session, the LLM decomposes each turn into multiple atomic fact bullets grouped by turn number. Each bullet is a complete, independent, self-contained statement preserving all specifics (names, dates, quantities). Like L_1, L_2 is generated on demand and cached. The prompt (Figure 6) instructs the model to split distinct facts into separate bullets and never merge multiple facts into one. Figure 13 shows a real L_2 output.

I.4 L_3: Raw Messages

Construction. No LLM processing is required. The verbatim user–assistant messages are stored directly, grouped by turn number. L_3 is materialized at ingestion time by simply formatting the original conversation into a structured Markdown file. The controller accesses L_3 via `read_raw` when exact wording or fine-grained details not preserved in higher layers are needed. Figure 14 shows a real L_3 output.

```
## Session 7 (date=2023/05/21)

Turn 1:
- The user asked how hydraulic fracturing
  affected groundwater quality in the
  Marcellus Shale region.
- The assistant stated that the Marcellus
  Shale region was located in the eastern
  United States.
- The assistant stated that some studies
  found higher levels of methane, benzene,
  and arsenic in groundwater near hydraulic
  fracturing wells.
- The assistant stated that other studies
  found no significant groundwater quality
  changes near hydraulic fracturing wells.
- The assistant stated that Pennsylvania
  required hydraulic fracturing companies
  to monitor groundwater quality at nearby
  wells before drilling.

Turn 2:
- The user stated distrust of hydraulic
  fracturing companies properly monitoring
  operations.
- The assistant stated that the EPA set and
  enforced regulations on hydraulic
  fracturing operations.
- The assistant stated that citizens could
  report suspected violations or file
  lawsuits against hydraulic fracturing
  companies.
```

Figure 13: L_2 output example: atomic fact bullets grouped by turn number.

Configuration	Avg. Acc. (%)
APDMem (full)	87.8
w/o draft answer	86.6
w/o Organized Notes	85.2
w/o Organized Notes & draft	83.6

Table 11: Generator ablation: effect of Organized Notes and draft answer on final accuracy (GPT-4.1, fixed retrieval).

J Generator-Side Ablation

To isolate the generator's contribution, we fix the retrieval results from a single run and vary only the generation strategy: whether evidence passes through the *Organized Notes Synthesizer* or is fed raw to the answer generator, and whether the controller's draft answer is included. Table 11 reports results.

The full system achieves 87.8%. Removing the draft answer reduces accuracy by 1.2 points, while removing Organized Notes causes a larger drop of 2.6 points. Without both, accuracy falls to 83.6% (-4.2), confirming that structured evidence synthesis and the controller's draft reasoning provide

```
## Session 0 (date=2023/02/15, id=f10be626)

### Turn 1

User:
I'm trying to plan my next trip and was
thinking about my past travel experiences.
Can you help me find some good destinations
for a solo trip?

Assistant:
I'd be happy to help you plan your next
solo adventure. To get started, can you
tell me:
1. What type of activities do you enjoy?
2. What's your budget for the trip?
3. How long do you have for the trip?
4. Are you open to exploring new destinations?
5. What type of accommodation do you prefer?
```

Figure 14: L_3 output example: verbatim user–assistant messages grouped by turn.

complementary benefits.

Organized Notes primarily helps categories requiring evidence aggregation: *multi-session synthesis* (+6.0 over raw) and *temporal reasoning* (+3.0), where the structured format, which organizes facts by relevance, enumerates items explicitly, and highlights contradictions, prevents the generator from anchoring on a subset of evidence. The draft answer primarily benefits single-session categories (*SS-User*: +2.9, *SS-Assistant*: +1.8), where the controller's focused reasoning about a specific session provides useful context for the generator.