ONDEVMEM: EFFICIENT LONG-TERM MEMORY FOR ON-DEVICE LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-term memory is essential for LLM-based agents to preserve and reuse information accumulated over extended interactions. Agent memory has advanced substantially under cloud-based settings, whereas fully on-device long-term memory remains at an early stage. Deploying existing memory pipelines unchanged on resource-constrained devices can compromise memory quality and incur substantial construction costs from frequent LLM-based semantic processing. Memory maintenance can further increase foreground inference latency by competing for limited local resources. To address these challenges, this paper introduces **OnDevMem**, a cloud-independent memory framework that separates evidence preservation, semantic indexing, and maintenance scheduling. Topic-aware Evidence Memory (EM) preserves source interactions and provenance, while locally generated Semantic Signatures (SS) provide complementary semantic signals for retrieval without replacing the original evidence. Semantic enrichment at topic granularity reduces memory-construction cost by processing coherent interaction segments jointly. A resource-adaptive controller coordinates signature generation and index updates with foreground demand and device conditions to reduce foreground latency. Experiments and ablations across two local deployments on LoCoMo show that OnDevMem improves judge accuracy by 11.6%–14.5% over the strongest evaluated baseline and reduces memory-construction time by 88.4%–92.5% compared with the fastest baseline across deployments.

1 INTRODUCTION

Long-term memory enables LLM agents to retain and reuse information across extended interactions. Recent agent-memory systems have progressed rapidly from retrieval-based access (Lewis et al., 2020) to stored interaction histories toward richer pipelines involving extraction, summarization, linking, and consolidation (Chhikara et al., 2025; Xu et al., 2025; Fang et al., 2026). Most existing systems are developed under cloud-capable settings with access to powerful language models and abundant computation. In practical deployments, privacy constraints may require interaction histories to remain on device, while unreliable connectivity and dependence on remote services can limit the availability of cloud inference (Wang et al., 2025; 2026). Fully local memory processing is therefore a practical requirement in many deployment scenarios, with memory construction, organization, retrieval, and downstream answering all executed on device. Despite rapid progress in agent memory, fully on-device long-term memory remains comparatively underexplored.

Directly moving existing memory pipelines to resource-constrained devices exposes several challenges. First, many memory systems rely heavily on LLM-based extraction, summarization, linking, or consolidation (Chhikara et al., 2025; Xu et al., 2025; Fang et al., 2026). When these semantic transformations are performed by smaller local models, generated memory representations may lose details or relationships needed by later queries, reducing downstream QA quality. Second, repeated local model calls make memory construction slow, and memory maintenance competes with foreground inference for the same CPU, GPU, and memory resources. When semantic processing overlaps with online answering, this contention can substantially increase QA latency. These challenges reflect a gap between existing memory pipelines and the requirements of on-device deployment, where memory quality and computational efficiency must be jointly considered.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To address these challenges, we introduce OnDevMem, a cloud-independent on-device long-term memory framework for LLM agents. The key idea is to redesign memory processing around the constraints of local execution by separating persistent evidence storage from resource-intensive semantic processing. OnDevMem introduces an EM that preserves original interaction records as the foundation for future reasoning, and a SS that provides compact semantic cues for efficient memory retrieval. This design allows source evidence to remain accessible throughout the memory lifecycle while enabling semantic organization to be performed according to available device resources.

Beyond memory representation, OnDevMem introduces a resource-adaptive maintenance mechanism to manage when semantic enrichment and index updates are executed. By monitoring runtime conditions and pending maintenance work, the controller schedules background memory processing to reduce interference with latency-sensitive foreground inference. Together, EM, SS, and adaptive maintenance enable OnDevMem to balance memory quality, construction efficiency, and runtime responsiveness under resource-constrained on-device deployment.

The contributions of this work are summarized as follows:

- We propose OnDevMem, a cloud-independent long-term memory framework for resource-constrained LLM agents, to address the degradation in memory quality, high memory-construction cost, and interference with foreground inference that arise in fully on-device deployment. This work advances long-term memory toward practical local execution without reliance on remote inference.
- The core architecture of OnDevMem combines topic-aware EM, locally generated SS, and dual-channel retrieval to improve memory accuracy while reducing memory-construction and retrieval cost. Resource-adaptive maintenance further coordinates semantic processing with foreground demand and device conditions, reducing interference with latency-sensitive inference and lowering foreground latency.
- Evaluations on LoCoMo across two fully local deployments show judge-accuracy gains of 11.6%–14.5% over the strongest baseline in each deployment and memory-construction time reductions of 88.4%–92.5% relative to the fastest baseline. Streaming experiments further demonstrate lower foreground latency under concurrent maintenance, and additional evaluation on LongMemEval further supports the effectiveness of OnDevMem across long-term memory benchmarks.

2 Related Work

Long-term memory for LLM agents. Prior work explores diverse mechanisms for retaining agent experience and making it useful in subsequent interactions. MemPrompt (Madaan et al., 2022) stores corrective feedback to guide future responses, Reflexion (Shinn et al., 2023) retains verbal reflections from task outcomes, and Generative Agents (Park et al., 2023) combines episodic retrieval with higher-level reflection. These approaches illustrate how stored experience can support behavior beyond the current interaction. For persistent conversational memory, MemoryBank (Zhong et al., 2024) introduces reinforcement and forgetting mechanisms, while MemGPT (Packer et al., 2023) manages information across memory tiers to extend the effective context available to an agent.

More recent systems emphasize the construction and evolution of organized memory representations. Mem0 (Chhikara et al., 2025) extracts salient facts and reconciles them with existing memories. A-Mem (Xu et al., 2025) constructs contextualized notes, links related memories, and updates their representations as new information arrives. StructMem (Xu et al., 2026a) introduces structured representations for long-horizon behavior, MemoryOS (Kang et al., 2025) coordinates storage and updating across a memory hierarchy, and MemGAS (Xu et al., 2026b) associates and selects memories at multiple granularities. Related approaches to document access use hierarchical summaries, graph-based associations, and compressed gists to locate relevant source information (Sarthi et al., 2024; Gutiérrez et al., 2024; Lee et al., 2024). A complementary research direction incorporates stored representations or latent memory into language models (Wu et al., 2022; Wang et al., 2023; 2024).

These studies establish the value of structured and persistent memory, but memory organization alone does not determine how reliably and efficiently a pipeline operates with a local model. In particular, generated abstractions and original interaction records serve different purposes: abstractions expose

useful retrieval cues, whereas source records retain details that later questions may require. OnDevMem makes this distinction explicit by preserving topic-aware source evidence and attaching locally generated SS as complementary retrieval representations. Source evidence remains accessible before semantic enrichment completes, allowing memory availability to be separated from the execution of semantic processing.

Efficient memory processing and on-device execution. Efficiency-oriented memory systems reduce the amount of semantic processing or move it outside the immediate response path. LightMem (Fang et al., 2026) introduces lightweight filtering, topic-aware organization, and sleep-time consolidation . Its separation of online processing from offline updates provides an important precedent for deferring memory operations. LangMem (LangChain, 2026) similarly supports background memory formation and consolidation. The small-model LightMem (Zhang et al., 2026) assigns online memory operations to specialized small language models and separates them from offline consolidation. Its retrieval pipeline combines coarse vector retrieval with semantic selection. OnDevMem instead generates reusable SS during memory construction and combines their relevance with source-text relevance, without language-model calls for query rewriting or reranking during retrieval.

Other efficiency techniques operate at the context or inference level. LLMLingua (Jiang et al., 2023) and LLMLingua-2 (Pan et al., 2024)compress input text to reduce context-processing costs, while StreamingLLM (Xiao et al., 2024) maintains selected attention states for efficient streaming inference. Activation-aware weight quantization (Lin et al., 2024) and flash-based weight loading (Alizadeh et al., 2024) reduce the hardware demands of local inference. These techniques complement memory-system design by making individual model operations less expensive.

Nevertheless, reducing operation cost or moving computation to the background does not by itself prevent contention when memory maintenance and foreground answering share a device. The remaining scheduling question is when semantic work can proceed without excessive foreground delay or prolonged maintenance deferral. OnDevMem addresses this trade-off through maintenance admission based on foreground activity, device resource pressure, and the amount and age of pending work. This extends the separation of online and background processing with explicit control over their competition for local resources.

3 METHOD

Overview. Figure 1 illustrates OnDevMem, an on-device memory framework with four components: **Evidence Construction**, **Semantic Enrichment**, **Dual-Channel Retrieval**, and **Adaptive Maintenance**. Evidence Construction segments interactions into topic-aware EM records that preserve source text and provenance. Semantic Enrichment uses a local language model to generate a SS for each record, providing additional cues for retrieval. Dual-Channel Retrieval combines source-text and signature relevance to select original evidence for answering. Adaptive Maintenance schedules signature generation and index updates according to foreground demand and device pressure. This organization allows the agent to retain new evidence independently of semantic processing and to enrich its retrieval representation as local resources permit.

3.1 EVIDENCE CONSTRUCTION

OnDevMem organizes interaction history into topic-aware EM records. Each record groups contiguous turns according to local topical continuity and retains their original content together with provenance metadata. This design preserves coherent conversational context within a source-grounded memory unit, while avoiding unnecessary loss of information during memory construction.

Prior work suggests that topic-granular processing can improve agent-memory construction by grouping related interactions before semantic processing (Fang et al., 2026). OnDevMem therefore segments each interaction session using local lexical statistics, without additional language-model inference or a learned segmentation model.

Given a session of ordered turns $h_1, \ldots, h_T$, each position after turn t is treated as a candidate boundary. Let L_t and R_t denote short neighboring windows on its two sides. A feature-hashing map

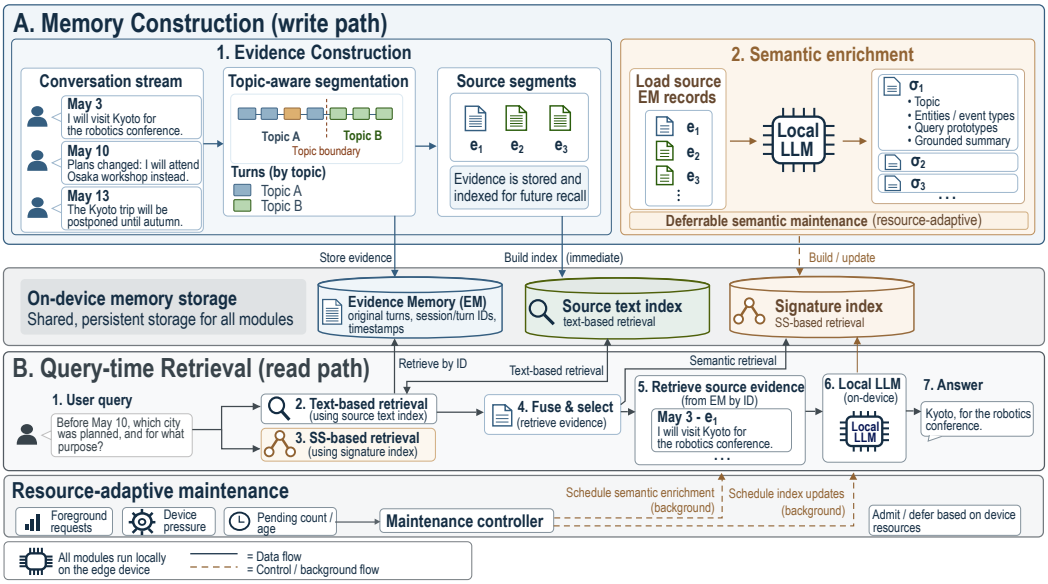


Figure 1: OnDevMem architecture on a single local device. Topic-aware segmentation forms source-preserving EM, and a local model generates SS for each record. Source-text and signature relevance jointly rank memories, while original EM records are provided to the local answer model. The maintenance controller uses foreground demand, device pressure, and pending-work status to admit or defer SS generation and signature-index updates; dashed arrows denote control flow and solid arrows denote data flow.

$\phi : h_j \mapsto \mathbb{R}^D$ provides inexpensive lexical representations, which are aggregated as

$$l_t = \sum_{j \in L_t} \phi(h_j), \qquad r_t = \sum_{j \in R_t} \phi(h_j). \tag{1}$$

The change in local context is measured by

$$\delta_t = 1 - \cos(l_t, r_t), \tag{2}$$

where a larger δ_t indicates a stronger potential topic transition. OnDevMem selects strong local peaks in δ_t using session-relative thresholds and applies turn- and token-count constraints to bound segment size.

Given the detected boundaries, OnDevMem constructs one EM record for each topic segment while retaining the original turn sequence without semantic compression. Let $0 = b_0 < b_1 < \cdots < b_N = T$ denote the detected topic boundaries. The i-th EM record is

$$e_i = (\xi_i, \eta_i), \qquad \xi_i = \mathrm{concat}\big(h_{b_{i-1}+1}, \ldots, h_{b_i}\big), \tag{3}$$

where ξ_i contains the original turns in the segment and η_i stores their source-turn identifiers together with available session and temporal metadata. This representation preserves coherent interaction context together with its provenance and serves as the source-grounded record used by later stages of the memory pipeline.

3.2 Semantic Enrichment

EM preserves the original interaction content, but leaves retrieval-relevant semantic cues implicit in the source text. To make these cues more explicit, OnDevMem augments each topic-aware EM record with a compact SS generated by a local language model.

For each EM record e_i, OnDevMem generates

$$(\tau_i, \mathcal{U}_i, \mathcal{V}_i, \zeta_i, \mathcal{P}_i) \leftarrow \mathrm{LLM}_\theta\big(\mathrm{Serialize}(e_i) \| p_{\mathrm{SS}}\big), \qquad \sigma_i = (\tau_i, \mathcal{U}_i, \mathcal{V}_i, \zeta_i, \mathcal{P}_i), \tag{4}$$

where LLM_θ denotes the local language model, p_{SS} is the structured generation prompt, and $\mathrm{Serialize}$ converts the source content and provenance of e_i into the model input. In the resulting signature, τ_i denotes the topic, $\mathcal{U}_i$ the entity set, $\mathcal{V}_i$ the event-type set, ζ_i a grounded summary, and $\mathcal{P}_i$ a small set of query prototypes.

These fields expose complementary semantic cues from the same source record. The topic, entities, and event types provide explicit semantic anchors, while the grounded summary captures relationships among them. The query prototypes serve as a supplementary field, providing a small number of plausible formulations through which the same evidence may later be referenced.

Using topic-aware EM records as the input to semantic enrichment provides a practical balance between semantic context and processing cost. Compared with turn-level inputs, topic-level records preserve more complete local context for extracting entities, events, and their relationships. Compared with session-level inputs, they limit unrelated content within the same generation context. Processing coherent segments jointly also reduces the number of local language-model calls required during memory construction.

3.3 Dual-Channel Retrieval

EM and SS provide complementary views of the same record. Source text preserves the original lexical context, while SS exposes higher-level semantic cues that may better align with differently phrased queries. OnDevMem therefore scores each record through both representations and fuses their relevance at the record level. Notably, although each SS is generated from its corresponding EM record and constrained to remain grounded in the source content, semantic generation may still omit details or produce imperfect abstractions. OnDevMem therefore retains the source-text channel and uses SS to refine retrieval rather than as a replacement for EM or a hard filter over source evidence.

For a user u, let $\mathcal{I}_u$ denote the identifiers of the user's EM records and let q be the current query. For each record $i \in \mathcal{I}_u$, the source and signature channels use $z_i^E = \xi_i$ and $z_i^S = d_i$, respectively, where d_i denotes the textual serialization of σ_i.

Each channel uses an independently fitted TF–IDF map v_c over its corresponding memory representation. The record and query vectors, together with their relevance score, are

$$\mathbf{x}_i^c = v_c(z_i^c), \qquad \mathbf{q}^c = v_c(q), \qquad s_i^c(q) = \cos\big(\mathbf{q}^c, \mathbf{x}_i^c\big), \qquad c \in \{E, S\}. \tag{5}$$

The source channel captures direct correspondence with the recorded interaction, whereas the signature channel matches the semantic descriptions constructed during enrichment.

Since the two channels are constructed from different textual representations, their similarity scores may have different scales. OnDevMem therefore normalizes scores independently within each channel:

$$\bar{s}_i^c(q) = \begin{cases} \dfrac{s_i^c(q) - m_c(q)}{M_c(q) - m_c(q)}, & M_c(q) - m_c(q) > \epsilon, \\ 0, & \text{otherwise}, \end{cases} \tag{6}$$

where

$$m_c(q) = \min_{j \in \mathcal{I}_u} s_j^c(q), \qquad M_c(q) = \max_{j \in \mathcal{I}_u} s_j^c(q),$$

and $\epsilon > 0$ avoids division by a near-zero score range.

The final relevance score is

$$s_i(q) = (1 - \alpha)\bar{s}_i^E(q) + \alpha \bar{s}_i^S(q), \tag{7}$$

where $\alpha \in [0, 1]$ controls the contribution of the signature channel. We use $\alpha = 0.25$ in the evaluated configuration, assigning greater weight to source-text relevance while allowing SS to refine the ranking. All records remain eligible until score fusion, so a weak signature match alone does not remove a source record from consideration.

Records are ranked by $s_i(q)$, and the top-K EM records are packed into the answer context in ranked order subject to the context budget. Only their original source text and provenance, rather than the generated signatures, are provided to the local answer model. Since SS are constructed during memory enrichment and reused across subsequent queries, dual-channel retrieval requires no additional language-model inference for query rewriting, candidate filtering, or reranking during retrieval.

3.4 Adaptive Maintenance

Although SS improve retrieval, their construction and signature-index updates require additional local computation and compete with foreground inference for limited CPU, GPU, and memory resources. EM becomes available independently of SS construction, allowing semantic enrichment and index updates to be deferred from the critical path of evidence ingestion. However, such deferral introduces a trade-off: eager maintenance may interfere with latency-sensitive QA, while excessive postponement can delay semantic organization and accumulate pending work. OnDevMem therefore adapts maintenance execution according to the current runtime state rather than eagerly processing every update.

At time t, the maintenance controller observes

$$s_t^{\text{maint}} = \big(F_t, R_t, D_t\big), \qquad a_t = \pi_{\text{maint}}\big(F_t, R_t, D_t\big), \tag{8}$$

where F_t represents foreground activity, R_t summarizes local resource pressure, and D_t represents maintenance debt. Foreground activity captures the demand from latency-sensitive online requests and serves as the primary signal for protecting responsiveness. Resource pressure summarizes the current CPU, GPU, and memory conditions and identifies execution periods with lower resource contention. Maintenance debt reflects the amount and age of pending semantic work, preventing deferred maintenance from being postponed indefinitely.

The controller prioritizes foreground responsiveness when deciding whether to admit semantic maintenance. When foreground activity is low, resource conditions guide maintenance toward lower-contention periods. Maintenance debt counterbalances such deferral: as pending work accumulates or ages, the controller gradually relaxes resource-related constraints to allow semantic organization to proceed. Together, these signals balance foreground responsiveness and long-term memory freshness under dynamic on-device conditions.

4 Experiments

The evaluation examines three questions: whether OnDevMem improves answer quality and memory-system efficiency in local deployments, how its memory representations affect retrieval and answering, and how adaptive maintenance balances foreground responsiveness against semantic-organization delay.

4.1 Experimental Setup

Datasets and deployments. We evaluate OnDevMem on LoCoMo (Maharana et al., 2024) with all ten conversations and 1,986 questions per method. An additional evaluation uses a fixed subset of 24 queries from LongMemEval (Wu et al., 2025) on Jetson Thor/Qwen3.5-35B. The subset evaluation reflects the substantial time required to construct memories and evaluate all methods locally on the full benchmark. Experiments are performed on two fully local deployments: NVIDIA Jetson AGX Orin with Qwen3.5-4B and Jetson Thor with Qwen3.5-35B. All methods construct memory locally using the same deployed model backend when language-model inference is required, and use the same local model with an identical QA prompt for downstream answering.

Baselines and metrics. We compare OnDevMem with LangMem (LangChain, 2026), Mem0 (Chhikara et al., 2025), A-Mem (Xu et al., 2025), and LightMem (Fang et al., 2026). All methods use the same retrieval setting with at most ten retrieved records and a 4,096-token evidence budget. Answer quality is evaluated using Token F1 and GPT-4o-mini judge accuracy. We measure memory construction time, retrieval latency, and streaming QA latency, where construction time reflects end-to-end memory building and retrieval latency measures memory access before answer generation.

Controlled and streaming evaluations. Architecture ablations are conducted on AGX Orin/Qwen3.5-4B using the same LoCoMo questions and retrieval configuration. For streaming evaluation, we replay LoCoMo conversations in their original turn order to simulate online memory operation. Each EM record becomes immediately available after its source interaction arrives, while semantic enrichment and index updates are executed as background maintenance. QA requests are injected only after their required evidence has appeared, forming a causal stream of 904 EM arrivals

Table 1: Fully local LoCoMo comparison on AGX Orin/Qwen3.5-4B and Jetson Thor/Qwen3.5-35B. Each method uses 1,986 questions per deployment. Acc. is evaluated by GPT-4o-mini, Token F1 is computed automatically, and build/retrieval costs denote memory construction and query-time retrieval overhead. Arrows indicate preferred directions; bold values denote the best results within each deployment.

Method	Acc. (%)↑	Token F1↑	Build (h)↓	Retrieval (ms)↓
AGX Orin / Qwen3.5-4B				
LangMem	16.5	0.148	65.04	38.15
Mem0	39.8	0.324	40.93	90.46
A-Mem	35.8	0.294	49.97	37.93
LightMem	33.9	0.286	20.57	118.01
OnDevMem	**54.3**	**0.454**	**2.394**	**4.268**
Jetson Thor / Qwen3.5-35B				
LangMem	25.7	0.232	21.63	67.10
Mem0	32.2	0.275	41.92	92.61
A-Mem	40.1	0.340	40.28	10.56
LightMem	41.2	0.346	19.96	148.26
OnDevMem	**52.8**	**0.451**	**1.493**	**2.155**

and 1,791 QA requests across 253 sessions. Foreground QA and maintenance share the same local inference service, and each policy is evaluated with three repeats.

4.2 Answer Quality and Memory-System Cost

Answer quality. Table 1 compares complete memory pipelines on LoCoMo under two fully local deployments. On AGX Orin/Qwen3.5-4B, OnDevMem achieves 54.3% judge accuracy and 0.454 Token F1, outperforming Mem0 by 14.5 percentage points in accuracy and 0.130 in Token F1. On Jetson Thor/Qwen3.5-35B, OnDevMem reaches 52.8% accuracy and 0.451 Token F1, improving over LightMem by 11.6 percentage points and 0.105 in Token F1. These consistent gains across different local model scales indicate that preserving source evidence while introducing semantic retrieval cues provides more reliable memory access for downstream answering.

Memory-system efficiency. OnDevMem substantially reduces both memory construction and retrieval cost. For the write path, OnDevMem completes memory construction in 2.394 hours on AGX Orin and 1.493 hours on Jetson Thor, reducing construction time by 88.4% and 92.5%, respectively, compared with the fastest baseline in each deployment. This efficiency results from the design of a lightweight memory construction pipeline that reduces the computational burden of semantic processing on resource-constrained devices.

On the 24-query LongMemEval subset, OnDevMem achieves 70.8% judge accuracy and 0.559 Token F1, exceeding A-Mem, the strongest baseline, by 8.3 percentage points and 0.089, respectively. It also achieves the lowest memory-construction time (5.296 hours). Appendix B reports the complete results.

A supplementary evaluation on 500 fixed LoCoMo questions further shows that OnDevMem retains the highest Token F1 under evidence-budget caps of 1,024, 2,048, and 3,072 tokens. Appendix D reports the complete results and realized context costs.

Overall, OnDevMem achieves consistent improvements in both answer quality and memory-system efficiency across different on-device deployments, showing its effectiveness for long-term memory management under resource constraints.

4.3 Design Ablations

Table 2 examines the main architectural choices of OnDevMem, including memory granularity, source-evidence preservation, and semantic enrichment for retrieval.

Table 2: Architecture ablations on AGX Orin/Qwen3.5-4B using 1,986 LoCoMo questions, at most ten retrieved records, and a 4,096-token evidence cap. Accuracy is evaluated by GPT-4o-mini; context reports mean evidence tokens and QA reports mean/P95 latency in seconds. Panel A varies EM granularity. Panel B fixes retrieved identifiers and changes the answer-context representation. Panel C changes ranking signals while returning source EM.

Configuration	Acc. (%)	Token F1	Context	QA mean/P95 (s)
A: Evidence-memory granularity				
Turn-pair EM + Full SS	50.15	0.4262	1,405	1.76 / 2.22
Topic-aware EM + Full SS	54.30	0.4540	2,828	2.65 / 3.70
Session-level EM + Full SS	54.28	0.4326	4,096	4.01 / 4.48
B: Source-evidence preservation				
Source EM context	54.30	0.4540	2,828	2.65 / 3.70
SS-only context (fixed retrieval)	38.67	0.3490	1,005	1.43 / 1.84
C: Semantic enrichment and retrieval representation				
Source EM only	52.87	0.4332	–	–
Source EM + Summary SS	53.52	0.4414	–	–
Source EM + Full SS	54.30	0.4540	–	–
SS-only ranking (source context)	49.45	0.4102	–	–

Evidence-memory granularity. To evaluate the effectiveness of topic-aware evidence construction, we compare topic-aware EM with turn-pair and session-level alternatives under the same semantic-enrichment procedure. As shown in Table 2(A), topic-aware EM achieves the best answer quality among the three granularities, while avoiding the substantially larger context and QA latency of session-level memory. This result supports topic-level segmentation as an effective granularity for evidence construction.

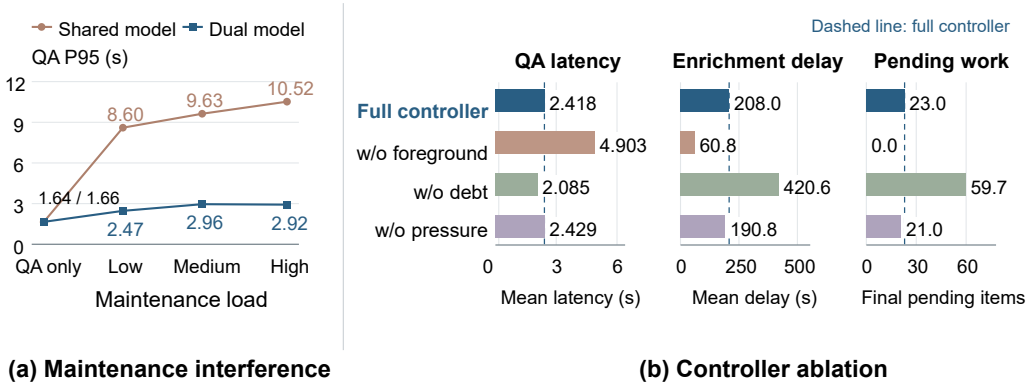
Source-evidence preservation. As shown in Panel B of Table 2, replacing the retrieved source EM with its Semantic Signature causes a large drop in Token F1, from 0.454 to 0.349, even though the resulting context is much shorter. This result highlights the importance of preserving the original interaction evidence for answering: semantic abstractions are useful for locating relevant memories, but should not replace the evidence ultimately provided to the answer model.

Semantic enrichment and dual-channel retrieval. As shown in Panel C of Table 2, semantic enrichment consistently improves retrieval over source-only ranking, with the full SS yielding the best result. However, using SS alone performs worse than combining it with source-text relevance. The two representations therefore play complementary roles: source text preserves direct lexical evidence, while SS introduces higher-level semantic cues that help match differently expressed queries.

4.4 Resource-Adaptive Memory Maintenance

Maintenance interference. To examine whether semantic maintenance interferes with foreground inference, we measure QA latency under increasing maintenance load with shared- and dual-model execution. As shown in Figure 2(a), Shared-model execution raises QA P95 latency to 6.41$\times$ the QA-only level. Using separate model instances reduces the interference, but foreground latency remains noticeably higher than the QA-only reference. These results show that concurrent semantic maintenance can substantially affect foreground responsiveness, motivating explicit control over when maintenance is executed.

Scheduling effectiveness. Table 3 compares Eager, Fixed-Matched, and Adaptive maintenance under the same streaming workload. Adaptive reduces mean QA latency by about 50% and P95 latency by about 28% relative to Eager, while maintaining comparable answer quality and keeping semantic organization available for most queries. This demonstrates that adaptive scheduling can move semantic work away from latency-sensitive foreground periods without preventing memory organization from progressing.



(a) Maintenance interference **(b) Controller ablation**

Figure 2: **Maintenance interference and scheduling trade-offs.** (a) Foreground QA P95 latency under increasing maintenance load with shared- and dual-model execution. (b) Mean QA latency, mean time from evidence arrival to completed enrichment, and final pending items under controller ablations. Results average three runs on AGX Orin, each with 1,791 queries and 904 evidence arrivals; dashed lines mark the full controller. Panels report separate experiments.

Table 3: Streaming maintenance on AGX Orin. Results are averaged over three repeats of a trace with 904 evidence arrivals and 1,791 QA requests. Accuracy is evaluated offline by GPT-4o-mini. QA latency includes foreground queueing and model service. Org. denotes semantic-organization availability at query time.

Policy	Acc. (%)	Token F1	QA mean (s)	QA P95 (s)	Org. (%)
Eager	37.06	0.3273	4.928	11.713	98.2
Fixed-Matched	36.57	0.3247	4.879	11.693	97.8
Adaptive	37.41	0.3355	2.418	8.472	94.8

Roles of the controller signals. Figure 2(b) further ablates the signals used by the maintenance controller. Removing foreground protection substantially increases QA latency, showing that it is the key mechanism for limiting interference. Removing maintenance-debt control reduces foreground latency but leads to much longer enrichment delay and a substantially larger pending backlog, confirming its role in preventing excessive deferral. Resource-pressure checks have a smaller effect under this workload. Overall, foreground protection and maintenance debt provide the main controls over responsiveness and maintenance progress.

5 Conclusion

Long-term memory is essential for LLM agents to maintain information across extended interactions, yet fully local deployment requires balancing memory quality, construction efficiency, and runtime responsiveness under constrained resources. In this work, we introduced OnDevMem, a cloud-independent on-device long-term memory framework that separates evidence preservation from semantic organization and adapts memory maintenance to runtime conditions. By decoupling persistent evidence availability from semantic processing, OnDevMem enables reliable memory access while reducing the computational burden of long-term memory management on local devices. Experiments on long-term memory benchmarks and real on-device deployments demonstrate that OnDevMem improves answer quality while substantially reducing memory construction cost and runtime interference compared with existing memory systems.

Future work will explore more adaptive memory organization mechanisms for dynamic and evolving environments. Beyond improving retrieval efficiency, future memory systems could further investigate how to build richer and more continuously updated semantic structures from accumulated experiences, enabling agents to better reason over long-term interactions and adapt their memory according to changing tasks and environments.

AI USE STATEMENT

During the preparation of this manuscript, the authors used generative AI tools for assistance with language polishing and improving the clarity of academic writing. All scientific ideas, methodologies, experiments, analyses, and conclusions were developed and verified by the authors.

REFERENCES

Keivan Alizadeh, Seyed Iman Mirzadeh, Dmitry Belenko, S. Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. LLM in a flash: Efficient Large Language Model Inference with Limited Memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12562–12584, 2024. URL https://aclanthology.org/2024.acl-long.678/.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. In *ECAI 2025*, 2025. doi: 10.3233/FAIA251160. URL https://journals.sagepub.com/doi/10.3233/FAIA251160.

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. LightMem: Lightweight and Efficient Memory-Augmented Generation. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/a05b72653ec5b473732129829ae04195-Abstract-Conference.html.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.nips.cc/paper_files/paper/2024/hash/6ddc001d07ca4f319af96a3024f6dbd1-Abstract-Conference.html.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376, 2023. doi: 10.18653/v1/2023.emnlp-main.825. URL https://aclanthology.org/2023.emnlp-main.825/.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory OS of AI Agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25961–25970, 2025. doi: 10.18653/v1/2025.emnlp-main.1318. URL https://aclanthology.org/2025.emnlp-main.1318/.

LangChain. LangMem: Core Concepts. Official documentation, 2026. URL https://langchain-ai.github.io/langmem/concepts/conceptual_guide/. Accessed September 15, 2026.

Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. A Human-Inspired Reading Agent with Gist Memory of Very Long Contexts. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 26396–26415, 2024. URL https://proceedings.mlr.press/v235/lee24c.html.

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. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, 2020. URL https://papers.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Proceedings of Machine Learning and Systems*, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html.

Aman Madaan, Niket Tandon, Peter Clark, and Yiming Yang. Memory-assisted prompt editing to improve GPT-3 after deployment. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022. URL https://aclanthology.org/2022.emnlp-main.183/.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating Very Long-Term Conversational Memory of LLM Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, 2024. URL https://aclanthology.org/2024.acl-long.747/.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems. arXiv preprint arXiv:2310.08560, 2023. URL https://arxiv.org/abs/2310.08560.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. URL https://aclanthology.org/2024.findings-acl.57/.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023. URL https://doi.org/10.1145/3586183.3606763.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/8a2acd174940dbca361a6398a4f9df91-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Bo Wang, Weiyi He, Shenglai Zeng, Zhen Xiang, Yue Xing, Jiliang Tang, and Pengfei He. Unveiling privacy risks in LLM agent memory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25241–25260, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1227. URL https://aclanthology.org/2025.acl-long.1227/.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting Language Models with Long-Term Memory. In *Advances in Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/ebd82705f44793b6f9ade5a669d0f0bf-Abstract-Conference.html.

Xunjie Wang, Jiacheng Shi, Zihan Zhao, Yang Yu, Zhichao Hua, and Jinyu Gu. TZ-LLM: Protecting on-device large language models with Arm TrustZone. In *Proceedings of the European Conference on Computer Systems (EuroSys '26)*, pp. 657–674, Edinburgh, Scotland, UK, 2026. Association for Computing Machinery. doi: 10.1145/3767295.3769334. URL https://doi.org/10.1145/3767295.3769334.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. MEMORYLLM: Towards Self-Updatable Large Language Models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL https://proceedings.mlr.press/v235/wang24s.html.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive

Memory. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/d813d324dbf0598bbdc9c8e79740ed01-Abstract-Conference.html.

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing Transformers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TrjbxzRcnf-.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5e5fd18f863cbe6d8ae392a93fd271c9-Abstract-Conference.html.

Buqiang Xu, Yijun Chen, Jizhan Fang, Ruobin Zhong, Yunzhi Yao, Yuqi Zhu, Lun Du, and Shumin Deng. StructMem: Structured Memory for Long-Horizon Behavior in LLMs. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 122–146, 2026a. URL https://aclanthology.org/2026.acl-short.12/.

Derong Xu, Yi Wen, Pengyue Jia, Yingyi Zhang, Wenlin Zhang, Yichao Wang, Huifeng Guo, Ruiming Tang, Xiangyu Zhao, Enhong Chen, and Tong Xu. From Single to Multi-Granularity: Toward Long-Term Memory Association and Selection of Conversational Agents. In *International Conference on Learning Representations*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/93e444b4a1a64815631c3c74977ee200-Abstract-Conference.html.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic Memory for LLM Agents. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

Jiaquan Zhang, Chaoning Zhang, Shuxu Chen, Zhenzhen Huang, Pengcheng Zheng, Zhicheng Wang, Ping Guo, Fan Mo, Sung-Ho Bae, Jie Zou, Jiwei Wei, and Yang Yang. Lightweight LLM Agent Memory with Small Language Models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12914–12929, 2026. URL https://aclanthology.org/2026.acl-long.588/.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing Large Language Models with Long-Term Memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 19724–19731, 2024. URL https://ojs.aaai.org/index.php/AAAI/article/view/29946.

A Evaluation Details

A.1 Independent Local Deployments

The LoCoMo evaluation uses all ten conversations and 1,986 questions per method on each deployment: NVIDIA Jetson AGX Orin with Qwen3.5-4B and NVIDIA Jetson Thor with Qwen3.5-35B.

Each method constructs its memory independently from raw interaction histories using the corresponding local model backend whenever language-model inference is required. Within each deployment, all methods use the same local answer model and QA prompt. The resulting memory state is frozen before retrieval and QA evaluation.

A.2 Retrieval and Architecture Controls

The main LoCoMo comparison and architecture ablations use at most ten retrieved memory records and a 4,096-token evidence budget. For OnDevMem, source-text and Semantic Signature relevance scores are independently normalized and fused with signature weight $\alpha = 0.25$.

Architecture ablations run on AGX Orin/Qwen3.5-4B using the same 1,986 LoCoMo questions. The granularity ablation varies the evidence-memory units while retaining the same semantic-enrichment procedure. The context-representation ablation fixes retrieved memory identifiers and changes the representation supplied to the answer model. The retrieval ablation varies the ranking signals while returning source EM as the answer context.

A.3 Quality Evaluation

Answer quality is evaluated using Token F1 and GPT-4o-mini judge accuracy. Token F1 measures lexical overlap between generated responses and reference answers. GPT-4o-mini assesses semantic correctness using the question, retrieved evidence, reference answer, and generated response under a common evaluation rubric.

The same evaluation procedure applies to all methods. Judging is performed offline and excluded from memory-construction, retrieval, and QA timing measurements.

A.4 Timing Measurement

Memory-construction time measures the wall-clock duration of building memory from raw interaction histories, including memory processing and index construction. Retrieval latency measures memory retrieval and ranking.

For the streaming evaluation, foreground QA latency includes queueing and answer-model service time. Queueing and service latency are also reported separately.

A.5 Streaming Evaluation

The streaming evaluation runs on AGX Orin and replays LoCoMo interactions in chronological order. It contains 253 sessions, 5,513 interactions, 904 EM arrivals, and 1,791 causally eligible QA requests over approximately 9,887 seconds. Session boundaries use a compressed interval of 20 seconds, and QA requests within the same session are separated by 2 seconds.

Each completed source segment becomes available as an EM record and enters the semantic-maintenance queue. QA requests are injected after their required evidence appears in the stream. Foreground QA and maintenance share the local inference service. Each maintenance policy is evaluated with three repeats.

We compare three maintenance policies:

- **Eager**: performs semantic maintenance immediately after each EM arrival.
- **Fixed-Matched**: periodically generates a fixed number of maintenance opportunities without considering foreground activity, resource conditions, or maintenance backlog.

Table 4: LongMemEval results on 24 queries using Jetson Thor/Qwen3.5-35B. Accuracy is evaluated by GPT-4o-mini. Build reports total memory-construction time for the subset, and retrieval reports mean latency per query. Bold indicates the best result in each column.

Method	Acc. (%) ↑	Token F1 ↑	Build (h) ↓	Retrieval (ms) ↓
LangMem	41.7	0.338	51.387	16.913
Mem0	54.2	0.433	47.751	102.695
A-Mem	62.5	0.470	76.237	17.047
LightMem	54.2	0.448	128.082	152.182
OnDevMem	**70.8**	**0.559**	**5.296**	**3.350**

- **Adaptive**: dynamically admits maintenance according to foreground activity, device resource pressure, and maintenance debt.

B Evaluation on LongMemEval

Experimental setup. Evaluating all methods on the full LongMemEval benchmark requires substantial time for local memory construction and question answering. The evaluation therefore uses a fixed subset of 24 queries. All methods run on Jetson Thor with Qwen3.5-35B and use the same query set, QA prompt, Top-10 retrieval limit, and 4,096-token evidence budget. Answer quality is measured using Token F1 and GPT-4o-mini judge accuracy. Memory construction and question answering execute locally, while judging is performed offline and excluded from timing measurements.

Results. Table 4 shows that OnDevMem achieves the highest judge accuracy and Token F1 among the evaluated methods. Compared with A-Mem, the strongest quality baseline, accuracy increases from 62.5% to 70.8%, and Token F1 increases from 0.470 to 0.559. OnDevMem also records the lowest construction and retrieval costs, requiring 5.296 hours to construct memory for the subset and 3.350 milliseconds per retrieval. These results complement the LoCoMo evaluation by showing higher answer quality and lower memory-system cost on the evaluated LongMemEval queries.

C Maintenance Controller and Scheduling Results

Workload and execution. The scheduling comparison and controller ablations use the AGX Orin streaming workload described in Appendix A, with three repeats per configuration.

Maintenance admission. The controller admits new maintenance operations according to foreground activity, resource pressure, and maintenance debt. Admission requires pending work and an available maintenance worker. In the shared-worker configuration, active or queued foreground requests block new maintenance admission. When foreground demand permits execution, resource-pressure checks determine whether device conditions allow maintenance to proceed. Maintenance debt, based on pending-work count and age, can override resource-pressure deferral.

Admitted operations execute sequentially and are non-preemptive. Foreground protection therefore controls the admission of new maintenance work; a QA request can still wait for a maintenance operation that has already started. Source EM remains retrievable while its semantic enrichment is pending.

Compared policies and ablations. Eager maintenance processes each EM record as soon as it arrives. Fixed-Matched provides maintenance opportunities on a fixed schedule without adapting to foreground activity, resource pressure, or pending-work state. Adaptive uses all three controller signals. Three ablations isolate their contributions: *w/o foreground* removes foreground-activity protection; *w/o debt* disables the pending-work override; and *w/o pressure* removes resource-pressure checks. The remaining controller rules stay unchanged in each ablation.

Foreground responsiveness. Table 5 shows that Adaptive reduces mean QA latency from 4.928 to 2.418 seconds relative to Eager, a reduction of 50.9%. QA P95 decreases from 11.713 to 8.472

Table 5: Answer quality and foreground latency in the AGX Orin streaming evaluation. Each policy uses three repeats of 1,791 QA requests. Accuracy and Token F1 report mean ± standard deviation across repeats. All latency columns are in seconds. QA measures foreground response latency, while Queue and Service report its queueing and model-service components.

Policy	Acc. (%)	Token F1	QA mean	QA P95	Queue P95	Service P95
Eager	37.06 ± 0.09	0.3273 ± 0.0009	4.928	11.713	10.241	1.928
Fixed-Matched	36.57 ± 0.06	0.3247 ± 0.0005	4.879	11.693	10.231	1.925
Adaptive	37.41 ± 0.11	0.3355 ± 0.0006	2.418	8.472	7.085	1.858
w/o foreground	36.93 ± 0.03	0.3270 ± 0.0004	4.903	11.624	10.235	1.934
w/o debt	38.27 ± 0.13	0.3361 ± 0.0020	2.085	6.615	5.422	1.848
w/o pressure	37.37 ± 0.12	0.3341 ± 0.0004	2.429	8.526	7.154	1.863

Table 6: Semantic-maintenance outcomes for the scheduling runs in Table 5, with 904 EM arrivals per run. Entries report averages across three repeats. Org. denotes semantic-organization availability at query time. Enrichment delay measures the time from evidence arrival to completed semantic enrichment. Pending and Completed report unfinished and finished maintenance items at the measurement endpoint.

Policy	Org. (%)	Delay mean (s)	Delay P95 (s)	Pending	Completed
Eager	98.2	54.9	124.8	0.0	904.0
Fixed-Matched	97.8	67.7	162.2	0.0	904.0
Adaptive	94.8	208.0	443.6	23.0	881.0
w/o foreground	98.0	60.8	131.1	0.0	904.0
w/o debt	90.6	420.6	838.3	59.7	844.3
w/o pressure	95.2	190.8	419.1	21.0	883.0

seconds, while observed accuracy and Token F1 increase slightly. Queue P95 decreases from 10.241 to 7.085 seconds, whereas Service P95 changes from 1.928 to 1.858 seconds. This pattern is consistent with reduced foreground queueing being the principal source of the latency improvement.

Removing foreground protection increases mean QA latency to 4.903 seconds, close to Eager's 4.928 seconds, and raises Queue P95 to 10.235 seconds. Foreground-aware admission is therefore the main mechanism for limiting maintenance interference in this workload.

Maintenance progress. Table 6 shows the corresponding cost of deferral. Adaptive completes an average of 881.0 out of 904 maintenance items, with 23.0 items pending and a mean enrichment delay of 208.0 seconds. Eager completes all items with a shorter enrichment delay but incurs substantially higher foreground latency.

Removing debt control further reduces mean QA latency to 2.085 seconds and yields higher observed accuracy (38.27% versus 37.41%). However, mean enrichment delay increases from 208.0 to 420.6 seconds, and pending work increases from 23.0 to 59.7 items. Debt control thus exchanges some foreground responsiveness for more timely semantic maintenance. Removing resource-pressure checks produces smaller changes: mean QA latency is 2.429 seconds and mean enrichment delay is 190.8 seconds. Foreground protection and debt control account for the larger observed effects in this workload.

D Context-Budget Evaluation

Experimental setup. The context-budget evaluation uses 500 fixed LoCoMo questions on AGX Orin/Qwen3.5-4B. All five methods use the same QA model, prompt, and tokenizer. Memory states and Top-10 retrieval rankings remain fixed across budget settings; only the maximum evidence context provided to the answer model changes. The evidence budgets are 1,024, 2,048, and 3,072 tokens, resulting in 7,500 QA responses across methods and budgets.

Table 7: Context-budget comparison on 500 fixed LoCoMo questions using AGX Orin/Qwen3.5-4B. Memory states and Top-10 retrieval rankings remain fixed while the evidence-budget cap varies. Evidence and Prompt report mean token counts; QA mean and QA P95 report latency in seconds. Bold indicates the highest Token F1 within each budget.

Budget	Method	Token F1	Evidence	Prompt	QA mean	QA P95
1,024	LangMem	0.1682	792.0	938.9	1.201	1.733
	Mem0	0.1663	352.0	499.0	0.568	1.091
	A-Mem	0.2412	939.9	1,087.0	1.207	1.760
	LightMem	0.2607	347.6	494.6	0.672	1.219
	OnDevMem	**0.3190**	860.9	1,008.0	1.348	1.750
2,048	LangMem	0.1815	1,738.1	1,885.1	1.678	2.470
	Mem0	0.1663	352.0	499.0	0.568	1.129
	A-Mem	0.2646	1,553.0	1,700.0	1.270	2.107
	LightMem	0.2607	347.6	494.6	0.676	1.194
	OnDevMem	**0.3349**	1,884.4	2,031.4	1.922	2.568
3,072	LangMem	0.1852	2,514.7	2,661.7	2.117	3.211
	Mem0	0.1663	352.0	499.0	0.547	1.099
	A-Mem	0.2664	1,553.7	1,700.8	1.184	2.200
	LightMem	0.2607	347.6	494.6	0.671	1.204
	OnDevMem	**0.3525**	2,670.4	2,817.4	2.288	3.191

Evidence records are packed in retrieval order as complete items. Records are not truncated, and packing stops when the next record exceeds the remaining budget rather than skipping it to include lower-ranked records. The budget applies to evidence text; total prompt length also includes the question and prompt instructions. The evaluation reports Token F1, realized evidence and prompt lengths, and QA latency.

Answer quality under constrained budgets. Table 7 shows that OnDevMem achieves the highest Token F1 at all three evidence-budget caps. At 1,024 tokens, OnDevMem reaches 0.3190, compared with 0.2607 for LightMem, the strongest baseline. At 2,048 and 3,072 tokens, its margins over the strongest baseline are 0.0703 and 0.0861, respectively. OnDevMem's Token F1 increases from 0.3190 to 0.3525 as the budget expands, showing that additional retrieved evidence improves answer quality in this evaluation.

Realized context and latency. The methods share the same evidence-budget caps, while realized input lengths depend on the sizes of their retrieved records. Mem0 and LightMem supply approximately 350 evidence tokens under all three caps, so larger budgets do not change their Token F1. A-Mem supplies approximately 1,554 evidence tokens at the two larger budgets and exhibits little further gain.

At the 1,024-token cap, OnDevMem achieves higher Token F1 than A-Mem while using fewer evidence tokens (860.9 versus 939.9). At larger caps, OnDevMem includes more evidence and attains higher answer quality, with a corresponding increase in QA latency. Its mean QA latency rises from 1.348 seconds at 1,024 tokens to 2.288 seconds at 3,072 tokens. The results characterize the quality–latency trade-off under a common evidence-budget constraint.