STOP WHEN MEMORY SUFFICES: EVIDENCE-CONDITIONED PROGRESSIVE EXECUTION FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon LLM agents must answer queries over histories whose supporting evidence may be localized or dispersed across interaction steps. Lightweight retrieval can miss decisive dependencies, whereas full memory reconstruction incurs unnecessary cost when the retrieved context already suffices. The required processing depth therefore depends on the evidence recovered for a query, not on query complexity alone. We introduce Router-Mem, an evidence-conditioned progressive execution framework that learns whether to stop after a shared retrieval prefix or continue with broader memory processing. To distinguish sufficiency from relevance, we construct supervision by retaining or removing supporting evidence and discarding candidate negatives that remain answerable. We train a lightweight router with binary label supervision and rationale-conditioned representation distillation to make a single-token termination decision without generating rationales at inference time. When continuation is selected, retrieval hits anchor block expansion, parallel memory analysis, and aggregation instead of restarting memory search from scratch. With DeepSeek-V4-Flash at a routing threshold of 0.5, Router-Mem achieves scores of 55.17% on AMA-Bench and 38.77% on BEAM, reducing online memory-processing time by 27.3% and 25.5% relative to full completion, respectively. Code is available at `https://anonymous.4open.science/r/Router-Mem-E0A5`.

1 INTRODUCTION

Long-horizon LLM agents accumulate conversations, tool calls, observations, and intermediate decisions while interacting with users and environments Yao et al. (2023); Schick et al. (2023); Shinn et al. (2023); Wang et al. (2024); Zhou et al. (2024); Liu et al. (2024b); Yang et al. (2024). Answering later queries requires recovering relevant memories together with the temporal and cross-step dependencies that explain past outcomes Li et al. (2024); Guo et al. (2024a). Although recurrent memory, sparse attention, and efficient attention mechanisms have extended context capacity Dai et al. (2019); Beltagy et al. (2020); Zaheer et al. (2020); Dao et al. (2022), processing the full history remains costly, and long-context models remain sensitive to evidence position, sequence length, and task complexity Liu et al. (2024a); Bai et al. (2024); Hsieh et al. (2024); Zhang et al. (2024). Agent memory must therefore balance selective access with the broader context needed for reliable answers.

Existing memory systems offer complementary trade-offs Zhang et al. (2025b); Hu et al. (2025); Du (2026) (Fig. 1a). Preprocessing-oriented methods, including Mem0, A-MEM, and G-Memory, consolidate or structure histories for efficient access Chhikara et al. (2025); Xu et al. (2025); Zhang et al. (2025a), but early compression may omit details needed by future queries. Query-time reconstruction methods such as GAM and E-mem recover broader evidence through search and analysis Yan et al. (2025); Wang et al. (2026), but incur unnecessary computation when initial retrieval already provides sufficient support. Some adaptive retrieval methods allocate computation based on predicted query complexity Jeong et al. (2024), yet query complexity alone cannot determine whether the evidence recovered in a particular execution is sufficient.

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.

The key principle is that *the need for further memory processing depends on the query and its currently retrieved evidence*. The same query may be answerable from one context but not another that omits a decisive step. Moreover, semantically relevant records need not form a complete evidence chain. Determining sufficiency can require reasoning about temporal updates, action–feedback relations, or conflicting records; an expensive online judgment could offset the savings from early termination. How can a memory system make this evidence-dependent decision cheaply and continue without discarding the work already performed during initial retrieval?

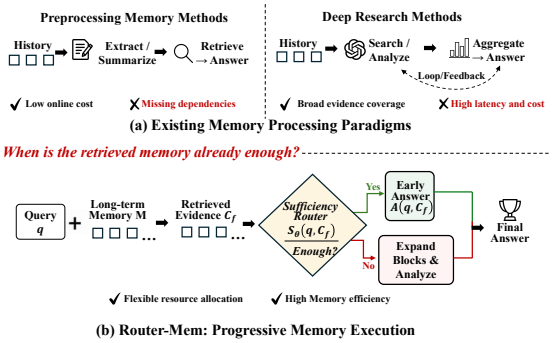


Figure 1: Existing memory paradigms vs. Router-Mem.

We introduce **Router-Mem**, an evidence-conditioned progressive execution framework for long-horizon agent memory (Fig. 1b). Every query first runs a shared low-cost retrieval prefix. A lightweight router evaluates the query and retrieved context to predict whether the current evidence supports early termination. If its sufficiency score meets a threshold, the system answers from the retrieved steps. Otherwise, retrieval hits locate source blocks and temporal neighborhoods for parallel memory analysis and aggregation. Both paths use the same answer model. The router therefore decides whether to continue a single memory-processing pipeline, rather than choosing between independent memory systems.

To learn sufficiency rather than topical relevance, we construct training contexts by retaining or removing annotated supporting evidence, adding distractors, and discarding candidate negatives that answer models can still solve. The router combines binary label supervision with rationale-conditioned representation distillation: a frozen teacher reads an evidence-judgment rationale during training, while the student reads only the query and context. At inference, the student makes a single-token decision without generating a rationale. The shared retriever is also trained on evidence-complete positives and evidence-incomplete negatives to improve the initial evidence available for both answering and routing.

On AMA-Bench Zhao et al. (2026) and BEAM Tavakoli et al. (2026), Router-Mem provides adjustable quality–latency trade-offs. With DeepSeek-V4-Flash at $\tau = 0.5$, it achieves scores of 55.17% and 38.77%, respectively, compared with 56.49% and 43.26% for Full Completion. It reduces online memory-processing time by 27.3% and 25.5%; the corresponding end-to-end latency reductions are 20.0% and 23.4%. Threshold sweeps and training ablations further characterize how evidence retrieval and sufficiency routing affect early termination, answer quality, and execution cost.

Our main contributions are as follows:

- We formulate long-horizon memory access as evidence-conditioned progressive execution, making early termination depend on the retrieved evidence state rather than relying on query complexity alone, thereby enabling adaptive computation allocation.
- We propose Router-Mem, combining a shared retrieval prefix, an evidence-supervised sufficiency router with rationale distillation, and retrieval-guided continuation. The framework reuses initial retrieval for both early answering and broader memory reconstruction.
- We evaluate Router-Mem on AMA-Bench and BEAM against memory baselines and fixed execution endpoints. Threshold sweeps and training ablations demonstrate controllable quality–latency trade-offs and characterize the contributions of retriever and router training.

2 Related Work

Retrieval-Augmented Generation. Retrieval-augmented generation (RAG) grounds LLMs in external knowledge and has become a standard paradigm for knowledge-intensive tasks Lewis et al. (2020). Some works like Adaptive-RAG, Stop-RAG, further selects among no retrieval, single-step retrieval, and iterative retrieval according to predicted query complexity Jeong et al. (2024); Park et al.

(2025); Li et al. (2026). However, this query-level routing does not account for the evidence actually recovered at runtime: a difficult query may be resolved by successful initial retrieval, whereas a seemingly simple query may still lack decisive evidence. Recent work also improves retrieval through structured representations. HippoRAG combines knowledge graphs with Personalized PageRank for associative and multi-hop retrieval Gutiérrez et al. (2024), while GraphRAG and LightRAG use graph-based indexing for global and hierarchical knowledge discovery Edge et al. (2024); Guo et al. (2024b). Despite improving retrieval planning and evidence organization, these methods do not directly determine whether the currently retrieved evidence is sufficient to terminate further computation.

Memory Systems for LLM Agents. Long-term agent memory systems aim to efficiently preserve and access past experiences. Preprocessing-based approaches construct compact memory representations before queries arrive. Mem0 extracts and consolidates salient conversational information Chhikara et al. (2025), A-MEM organizes experiences as evolving structured notes Xu et al. (2025), and G-Memory builds hierarchical graphs for multi-agent interactions Zhang et al. (2025a). ReasoningBank further distills reusable reasoning strategies from agent trajectories Ouyang et al. (2026). These methods reduce retrieval cost but may lose fine-grained temporal and causal dependencies. In contrast, query-time reconstruction methods retain broader histories and perform deeper reasoning when queries arrive. GAM employs researcher agents to construct query-specific contexts Yan et al. (2025), while E-mem uses episodic retrieval and multi-agent analysis for context reconstruction Wang et al. (2026). Although these methods improve evidence coverage, they may introduce unnecessary computation when lightweight retrieval already provides sufficient evidence.

Our work connects these two regimes through a shared retrieval prefix and an evidence-sufficiency router, which terminates early when the current context supports an answer and preserves deeper processing for queries that require it.

3 METHOD

In this section we systematically presents the design of Router-Mem including training process and system pipeline. More details can be found in **Appendix B**.

3.1 PROGRESSIVE MEMORY EXECUTION

Long-horizon agents continuously accumulate heterogeneous memories, including conversations, intermediate decisions, tool calls, environment observations, and execution feedback. Queries over such memories require different levels of processing. Some queries can be answered from a small set of locally relevant memories recovered by lightweight retrieval. Others depend on evidence that is dispersed across time, execution steps, or interaction trajectories, and therefore require broader memory expansion and aggregation. Applying only lightweight retrieval to every query may miss such dependencies, whereas executing the full memory-processing pipeline for every query introduces unnecessary latency and inference cost.

We therefore organize memory access as a progressive execution process. Every query first performs a shared low-cost retrieval prefix. If the retrieved context already contains sufficient evidence, the system directly generates an answer and skips the remaining computation. Otherwise, it continues with broader memory processing. The fast and slow paths are coupled through the shared retrieval prefix, as the expensive stage continues from the retrieved evidence and reuses its localization signals for broader memory processing.

Determining whether the retrieved context is sufficient is itself non-trivial. A direct solution is to prompt a general-purpose LLM to inspect the query and retrieved memories and explicitly reason about whether further processing is needed. However, a reliable judgment may require checking for missing evidence, temporal updates, action–feedback relations, and conflicting records. Although the final output is only a binary decision, the underlying reasoning can be complex. Performing such deliberation online often requires a capable model, a long context prefill, and additional reasoning tokens. This cost is undesirable because the router is introduced precisely to avoid unnecessary computation. An expensive routing decision may substantially reduce the efficiency gain obtained from early termination.

To address this issue, we train a lightweight evidence-sufficiency router that directly predicts whether the current retrieval evidence state supports early termination. Given a query q and an ordered memory $M = (z_1, \ldots, z_T)$, the shared retriever first returns

$$C_f = R_f(q, M), \tag{1}$$

where C_f is the initially retrieved memory context. We define the context-dependent termination label as

$$y(q, C_f) = \mathbf{1}\left[C_f \text{ provides sufficient evidence to answer } q\right], \tag{2}$$

and train the router to estimate

$$s_\theta(q, C_f) = P_\theta\left(y = 1 \mid q, C_f\right). \tag{3}$$

Importantly, the decision is conditioned on the execution state (q, C_f) rather than on the query alone. The same query may terminate after one retrieval result but require further processing after another result that omits a decisive memory step. During training, we use evidence-level supervision and rationale distillation to transfer the required evidence judgment into the lightweight router. At inference time, the router reads only (q, C_f) and produces a single-token termination decision.

3.2 Evidence-Sufficiency Data Construction

Training the router requires supervision that distinguishes evidence sufficiency from semantic relevance (as shown in Fig. 2). We collect long-horizon agent trajectories from diverse interaction settings, including tool use, software engineering, web navigation, games, and long conversations. These training sources are disjoint from the downstream evaluation benchmarks, while preserving similar interaction patterns such as multi-step actions, environment feedback, and temporally dispersed evidence.

For each trajectory, we construct and verify a query–answer–evidence tuple (q_i, a_i, E_i), where E_i contains the memory steps that support the reference answer a_i. We then apply evidence intervention to produce contexts with different sufficiency labels. A positive context retains all required evidence and mixes it with distractor steps:

$$C_i^+ = E_i \cup D_i, \tag{4}$$

where D_i contains randomly sampled memory steps from the same trajectory. Since the complete supporting evidence is preserved, C_i^+ is labeled as sufficient.

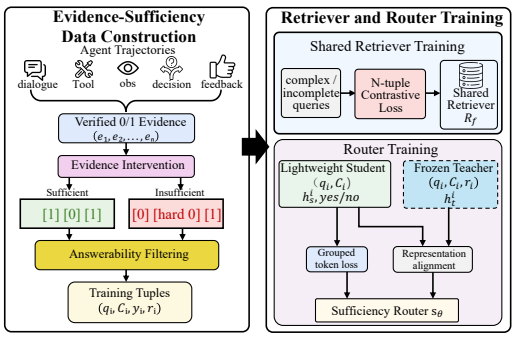


Figure 2: Training process of Router-Mem.

A negative context removes part or all of the required evidence and fills the remaining context with semantically related hard distractors:

$$C_i^- = \left(E_i \setminus E_i^{\text{drop}}\right) \cup D_i^{\text{hard}}. \tag{5}$$

The hard distractors may share entities, tools, or nearby events with the query, but do not preserve the complete evidence chain. These examples teach the router that relevance alone does not imply answerability.

Evidence removal does not always make a context insufficient, because redundant or alternative support may remain. We therefore evaluate each candidate negative with multiple answer models and discard it if any model still produces a correct answer. For every retained sample, a teacher model generates a rationale that explains which evidence is present and which decisive information is missing. The resulting dataset consists of tuples (q_i, C_i, y_i, r_i), where $y_i \in \{\text{yes}, \text{no}\}$ is the

sufficiency label and r_i is the teacher rationale. We split the data by trajectory to prevent related questions, neighboring steps, and context variants from leaking across training and validation sets.

3.3 Training Process

Given a constructed sample (q_i, C_i, y_i, r_i), we form the student input as

$$x_i = \text{Prompt}(q_i, C_i). \tag{6}$$

The router retains the causal language-model interface and predicts the sufficiency label through its next-token distribution. Let V_{yes} and V_{no} denote token groups corresponding to the normalized outputs "yes" and "no." We optimize the grouped-token classification loss

$$\mathcal{L}^i_{\text{cls}} = -\log \sum_{v \in V_{y_i}} P_\theta(v \mid x_i). \tag{7}$$

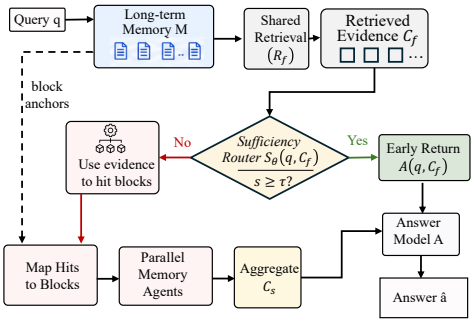


Figure 3: System pipeline of Router-Mem.

The binary label specifies the desired decision but provides limited supervision about the underlying evidence judgment. We therefore use rationale-conditioned representation distillation. A frozen teacher additionally reads the rationale r_i, while the student observes only (q_i, C_i). Let h^i_s denote the student representation before the decision token, and let h^i_t denote the teacher representation after processing the rationale. We align them at a selected intermediate layer using

$$\mathcal{L}^i_{\text{rep}} = 1 - \frac{(h^i_s)^\top h^i_t}{\|h^i_s\|_2 \|h^i_t\|_2}. \tag{8}$$

The final training objective is

$$\mathcal{L}_{\text{router}} = \mathcal{L}_{\text{cls}} + \lambda_{\text{rep}} s_0 \mathcal{L}_{\text{rep}}, \tag{9}$$

where λ_{rep} controls the contribution of representation distillation and s_0 calibrates the scales of the two losses. The teacher and rationale are used only during training. At inference time, the student reads (q, C_f) and produces a single decision token without generating an explanation.

Shared retriever training. We train the shared retriever with the same QA–evidence supervision. Evidence-complete memory views are treated as positives, while semantically related but incomplete views are used as negatives. Using an L2-normalized encoder $e(\cdot)$, we optimize a multi-positive contrastive objective:

$$\mathcal{L}^i_{\text{ret}} = -\frac{1}{|P_i|} \sum_{p \in P_i} \log \frac{\exp\left(e(q_i)^\top e(p)/T\right)}{\sum_{c \in \mathcal{C}_i} \exp\left(e(q_i)^\top e(c)/T\right)}. \tag{10}$$

This objective encourages the shared prefix to retrieve answer-supporting evidence.

3.4 PROGRESSIVE MEMORY SYSTEM

After training the retriever and termination router, we integrate them into a progressive memory execution system. The system first constructs aligned retrieval and processing views of the same memory (as shown in Fig. 3). It then performs a shared retrieval step for every query and invokes broader memory processing only when the retrieved evidence is predicted to be insufficient.

Aligned memory views. Given an ordered memory $M = (z_1, \ldots, z_T)$, we construct two aligned views. The retrieval view indexes individual memory steps and stores their corresponding step and block identifiers. The processing view groups temporally adjacent steps into overlapping memory blocks.

Algorithm 1: Progressive Memory Execution

Require: Query q, memory M, threshold τ
Ensure: Answer $\hat{a}$
$(C_f, H_f) \leftarrow R_f(q, M)$ ▷ Shared retrieval prefix
if C_f fits the router input budget **then**
  $s \leftarrow s_\theta(q, C_f)$ ▷ Evidence-sufficiency score
  if $s \geq \tau$ **then**
    $\hat{a} \leftarrow A(q, \mathrm{Order}(C_f))$
    return $\hat{a}$ ▷ Optimistic early return
  end if
end if
$\mathcal{B}_f \leftarrow \mathrm{MapToBlocks}(H_f)$
$\mathcal{B}_s \leftarrow \mathrm{SelectBlocks}(\mathcal{B}_f, M)$
for all $b \in \mathcal{B}_s$ **in parallel do**
  $u_b \leftarrow \mathrm{MemAgent}(q, b)$
end for
$C_s \leftarrow \mathrm{Aggregate}(\{u_b : b \in \mathcal{B}_s\})$
$\hat{a} \leftarrow A(q, C_s)$
return $\hat{a}$

This alignment allows each retrieval hit to directly locate the memory blocks used by subsequent processing. The two views therefore support different stages of one execution process, rather than two independent memory systems.

Shared retrieval and termination. For a query q, the shared retriever first returns an initial context C_f and its retrieval hits H_f:

$$C_f, H_f = R_f(q, M). \tag{11}$$

The router evaluates the sufficiency of C_f through its next-token distribution. We aggregate the probability assigned to the "yes" and "no" token groups:

$$p_{\mathrm{yes}} = \sum_{v \in V_{\mathrm{yes}}} P_\theta(v \mid q, C_f), \qquad p_{\mathrm{no}} = \sum_{v \in V_{\mathrm{no}}} P_\theta(v \mid q, C_f), \tag{12}$$

and compute the normalized termination score

$$s(q, C_f) = \frac{p_{\mathrm{yes}}}{p_{\mathrm{yes}} + p_{\mathrm{no}}}. \tag{13}$$

Given a threshold τ, the system follows

$$\pi_\tau(q, C_f) = \begin{cases} \mathrm{terminate}, & s(q, C_f) \geq \tau, \\ \mathrm{continue}, & s(q, C_f) < \tau. \end{cases} \tag{14}$$

Varying τ yields different empirical trade-offs between answer quality and execution cost. If the retrieved context exceeds the router input budget, the system conservatively selects continuation.

Early return and execution continuation. If the termination condition is satisfied, the retrieved steps are restored to their original temporal order and passed to the answer model:

$$\hat{a}_f = A(q, C_f). \tag{15}$$

This branch is an early return from the full memory execution, rather than the output of an independently selected fast system.

Otherwise, the system reuses the retrieval hits to continue the same execution process. Let $\mathcal{B}_f$ denote the blocks associated with H_f. We expand these anchors into a bounded set of processing blocks:

$$\mathcal{B}_s = \mathrm{SelectBlocks}(\mathcal{B}_f, M). \tag{16}$$

The selected blocks include the retrieved anchor regions and their temporal neighborhoods, which helps recover local action–feedback and cross-step dependencies. Each block is analyzed independently by a memory agent:

$$u_b = \mathrm{MemAgent}(q, b), \qquad b \in \mathcal{B}_s. \tag{17}$$

Table 1: Results on AMA-Bench with two backbone LLMs: DeepSeek-V4-Flash and Qwen3.5-35B-A3B.

	DEEPSEEK-V4-FLASH					QWEN3.5-35B-A3B				
Method	Score (%)	Early (%)	T_{mem} (s)	T_{e2e} (s)	T_{build} (s)	Score (%)	Early (%)	T_{mem} (s)	T_{e2e} (s)	T_{build} (s)
External Memory Baselines										
RAG	40.46	–	0.057	2.49	10.15	45.21	–	0.059	4.72	9.88
LightMem	35.34	–	0.058	2.30	1157.78	36.58	–	0.055	2.49	6531.13
Mem0	35.90	–	0.159	3.50	46.12	43.96	–	0.156	2.61	51.37
Long Context	47.08	–	3.595	5.18	0.001	53.95	–	20.520	22.80	0.002
GAM	40.21	–	22.450	25.289	123.57	43.96	–	24.497	28.40	141.52
Fixed Execution Endpoints										
Fast Prefix Only	44.91	100.0	0.054	3.70	5.07	48.09	100.0	0.054	3.63	4.56
Full Completion (Slow)	56.49	0.0	11.341	14.47	5.07	56.41	0.0	15.816	20.28	4.66
Router-Mem: Evidence-Conditioned Early Termination										
Router-Mem ($\tau = 0.1$)	51.48	53.7	5.608	9.21	4.93	52.50	54.6	8.414	12.80	4.57
Router-Mem ($\tau = 0.3$)	52.68	38.5	7.215	10.53	4.90	53.65	39.2	9.351	13.10	4.65
Router-Mem ($\tau = 0.5$)	55.17	30.0	8.248	11.57	5.08	55.06	30.2	10.581	14.49	4.60
Router-Mem ($\tau = 0.7$)	54.73	22.1	9.045	12.26	4.98	55.52	22.3	13.888	18.67	4.71
Router-Mem ($\tau = 0.9$)	55.89	12.3	10.273	13.51	4.94	55.52	12.5	14.423	18.96	4.61

An aggregator combines the block-level results into a consolidated context:

$$C_s = \text{Aggregate}\left(\{u_b : b \in \mathcal{B}_s\}\right), \qquad \hat{a}_s = A(q, C_s). \tag{18}$$

The continuation stage therefore preserves and extends the computation performed by the shared retrieval prefix. It is invoked only when the current evidence does not support early termination.

4 EXPERIMENT

We evaluate Router-Mem against state-of-the-art memory systems for LLM agents, focusing on effectiveness, routing efficiency, and robustness. Our experiments investigate three key research questions:

RQ1: Comparative Effectiveness. Does Router-Mem provide advantages over existing memory mechanisms in terms of answer quality, inference latency, and overall efficiency?

RQ2: Quality–Latency Trade-off. Can Router-Mem achieve a favorable balance between memory answer performance and inference latency, and how does the routing threshold control this trade-off?

RQ3: Training Effectiveness. Do the proposed router training and embedding training improve retrieval quality, routing decisions, and end-to-end system performance?

4.1 EXPERIMENTAL SETUP.

Dataset. We evaluate Router-Mem on two representative long-term memory benchmarks for LLM agents: AMA-Bench Zhao et al. (2026) and BEAM Tavakoli et al. (2026). AMA-Bench is designed to evaluate memory capabilities in long-horizon agentic scenarios, covering diverse interactions where agents must retrieve and reason over previously observed experiences. It emphasizes accurate memory access under complex multi-step dependencies. BEAM focuses on extremely long-context memory understanding and evaluates whether agents can effectively recover relevant information from large-scale conversational histories. Together, these benchmarks cover complementary challenges: AMA-Bench tests structured agent memory reasoning, while BEAM stresses scalability under massive context lengths. They provide a comprehensive evaluation of both memory effectiveness and efficient execution.

Settings. We compare Router-Mem with representative memory baselines, including standard RAG Lewis et al. (2020), compact memory systems such as LightMem Fang et al. (2026) and Mem0 Chhikara et al. (2025), Long Context, and deep query-time memory reconstruction methods GAM Yan et al. (2025). For evaluation, we use DeepSeek-V4-Flash as the backbone model and report answer quality measured by the official LLM judge provided by each benchmark. We evaluate both efficiency and effectiveness by reporting memory construction time, retrieval latency, and end-to-end inference latency. Further experiments analyze the impact of routing thresholds τ, router training components, and embedding model choices to evaluate the robustness and scalability of Router-Mem. We also evaluate the E2E token cost of Router-mem against existing agentic memory baselines in **Appendix D**.

Table 2: End-to-end results on BEAM with Model DeepSeek-V4-Flash.

Method	Score (%)	Early Term. (%)	T_{mem} (s)	T_{e2e} (s)	T_{build} (s)
External Memory Baselines					
RAG	30.29	–	0.094	1.66	217.01
LightMem	27.12	–	0.058	1.55	1174.31
Mem0	22.09	–	0.151	1.75	982.20
Long Context	38.01	–	14.304	17.08	0.05
GAM	35.23	–	25.817	28.435	1334.78
Fixed Execution Endpoints					
Fast Prefix Only	30.02	100.0	0.054	2.51	31.81
Full Completion (Slow)	43.26	0.0	14.611	18.43	31.90
Router-Mem: Evidence-Conditioned Early Termination					
Router-Mem ($\tau = 0.1$)	33.94	56.9	5.675	8.51	32.21
Router-Mem ($\tau = 0.3$)	36.53	39.4	8.781	11.72	31.84
Router-Mem ($\tau = 0.5$)	38.77	26.6	10.883	14.11	31.88
Router-Mem ($\tau = 0.7$)	39.81	16.7	12.512	16.18	32.01
Router-Mem ($\tau = 0.9$)	42.09	8.0	13.422	16.93	31.86

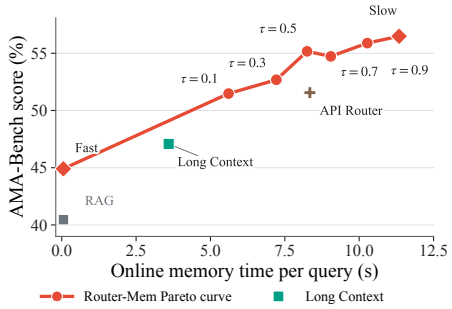


Figure 4: Quality–latency under different routing thresholds. API router means use API for routing with model DeepSeek-V4-flash.

Table 3: Router training comparison on AMA-Bench and BEAM. ET denotes the early-termination rate.

		AMA-Bench			BEAM		
τ	Router	Score (%)	ET (%)	T_{e2e} (s)	Score (%)	ET (%)	T_{e2e} (s)
0.3	Base	55.63	8.0	13.68	29.71	90.5	4.15
	Trained	52.68	38.5	10.53	36.53	39.4	11.72
0.5	Base	55.52	4.9	14.21	29.62	86.5	5.15
	Trained	55.17	30.0	11.57	38.77	26.6	14.11
0.7	Base	55.79	2.2	14.39	31.78	80.5	5.67
	Trained	54.73	22.1	12.26	39.81	16.7	16.18

4.2 Main Results.

We compare Router-Mem with representative memory mechanisms from three categories. RAG serves as a lightweight retrieval baseline that directly retrieves relevant contexts without persistent memory modeling. LightMem and Mem0 represent compact memory systems that extract and organize historical information into efficient memory representations. Long Context provides a full-context baseline that directly exposes the entire history to the language model. We further compare against two execution endpoints: Fast Prefix Only, which always terminates after the shared retrieval stage, and Full Completion (Slow), which always performs complete memory reconstruction. These endpoints characterize the lower-cost and upper-performance boundaries of progressive memory execution.

Results on AMA-Bench. Table 1 shows that Router-Mem consistently achieves a favorable quality–latency trade-off between the fast and slow endpoints. The Fast Prefix Only baseline achieves the lowest memory latency (0.054s) but suffers from incomplete evidence coverage, obtaining 44.91% score. In contrast, Full Completion reaches the highest score of 56.49% with substantially larger latency. By adaptively routing queries, Router-Mem approaches the performance of full completion while avoiding unnecessary deep memory processing. For example, Router-Mem ($\tau = 0.5$) achieves 55.17% score with only 8.25s memory latency, reducing the cost by 27.3% compared with full completion. Different thresholds provide controllable operating points, demonstrating that the learned router effectively balances answer quality and execution efficiency.

Results on BEAM. Results on BEAM further demonstrate the scalability of Router-Mem under extremely long conversational histories. The Fast Prefix Only baseline obtains only 30.02% score, while Full Completion improves performance to 43.26% at the cost of 14.61s memory latency. Router-Mem bridges this gap by allocating deeper reasoning only when necessary. With $\tau = 0.7$, it achieves 39.81% score while reducing memory latency to 12.51s, and with $\tau = 0.9$, it approaches the full completion performance (42.09%) with lower inference cost. Across thresholds, Router-Mem forms a smooth quality–cost frontier, validating that evidence-conditioned routing enables adaptive computation allocation for long-term agent memory.

Table 4: Effect of embedding training on AMA-Bench.

Setting	Embedding	Score (%)	Fast hit (%)	$\mathbf{T}_{\mathrm{mem}}$ (s)	$\mathbf{T}_{\mathrm{e2e}}$ (s)
Fast-only	Base	42.00	100.0	0.055	3.41
Fast-only	Trained	44.91	100.0	0.054	3.70
Router ($\tau = 0.5$)	Base	54.79	21.8	9.17	12.16
Router ($\tau = 0.5$)	Trained	55.17	30.0	8.25	11.57
Slow-only	Base	55.73	0.0	11.29	14.32
Slow-only	Trained	56.49	0.0	11.34	14.47

Table 5: Effect of jointly training the router and retriever on AMA-Bench.

τ	Training	Score (%)	Early Term. (%)	$\mathbf{T}_{\mathrm{mem}}$ (s)	$\mathbf{T}_{\mathrm{e2e}}$ (s)
0.3	None	54.48	9.4	10.352	13.453
	Joint	52.68	38.5	7.215	10.534
0.5	None	53.96	6.1	10.842	13.987
	Joint	55.17	30.0	8.248	11.570
0.7	None	56.25	3.5	11.073	14.219
	Joint	54.73	22.1	9.045	12.260

4.3 Ablation Study

Memory Performance and Latency Trade-off. Figure 4 illustrates how the routing threshold controls the quality–latency trade-off on AMA-Bench. Increasing τ sends more queries to deeper memory processing, generally improving quality at the cost of online memory time. Among the operating points, $\tau = 0.5$ provides a favorable Pareto-efficient trade-off and outperforms the large model API Router baseline at comparable memory time. It achieves 55.17%, only 1.32 points below Full Completion (56.49%), while reducing T_{mem} from 11.341s to 8.248s and T_{e2e} from 14.47s to 11.57s, corresponding to 27.3% and 20.0% reductions. Thus, it retains most of the full-memory accuracy while avoiding substantial computational overhead.

Effect of Router Training. As shown in Table 3, router training calibrates the fast–slow decision boundary. On AMA-Bench at $\tau = 0.5$, it raises early termination from 4.9% to 30.0% and reduces T_{e2e} from 14.21s to 11.57s, while preserving a comparable score (55.52% vs. 55.17%). On BEAM, it corrects the base router's over-confidence in the fast path, reducing early termination from 86.5% to 26.6% and improving the score from 29.62% to 38.77%. The trained router also responds consistently to τ, unlike the base router's opposite fixed-endpoint biases across the two benchmarks, making threshold adjustment effective.

Effect of Embedding Training. Table 4 shows that the trained embedding improves the Fast-only score from 42.00 to 44.91 without changing memory latency. At $\tau = 0.5$, it raises the score from 54.79 to 55.17 and the fast-hit rate from 21.8% to 30.0%, while reducing T_{mem} from 9.17s to 8.25s and T_{e2e} from 12.16s to 11.57s. The Slow-only endpoint also gains 0.76 points with nearly unchanged runtime, indicating that embedding training improves evidence quality and routing efficiency rather than increasing computation.

Effect of Joint Training. Table 5 shows that without training, early termination remains at 3.5%–9.4% and τ provides little control over computation. Joint training expands this range to 22.1%–38.5% while reducing memory and end-to-end latency with comparable answer quality. At $\tau = 0.5$, it raises the score from 53.96% to 55.17% and early termination from 6.1% to 30.0%, while reducing T_{mem} from 10.842s to 8.248s. Thus, the learned components jointly calibrate retrieval quality and termination confidence, making τ an effective quality–latency control.

5 Conclusion

In this paper, we propose Router-Mem to address the challenge of efficiently accessing long-horizon memories in LLM agents, where existing methods often trade off between low-cost retrieval and expensive deep memory reasoning. Router-Mem is an evidence-conditioned progressive memory execution framework that learns when the current retrieved context is sufficient to terminate further computation. It first performs a shared retrieval prefix and employs a lightweight sufficiency router to make an early-termination decision. In particular, the shared prefix couples the fast and slow paths, allowing the system to preserve retrieved evidence and avoid restarting memory search when deeper processing is required. When evidence is insufficient, it reuses retrieval anchors to guide memory expansion and deeper evidence aggregation. Experiments on AMA-Bench and BEAM demonstrate that Router-Mem achieves strong answer quality while substantially reducing inference cost. We believe our approach provides a practical direction toward adaptive memory systems that allocate reasoning resources according to the actual evidence requirements of each query.

REPRODUCIBILITY STATEMENT

We provide the necessary details to reproduce our results throughout the paper and Appendix. The main text describes the router-mem framework. The appendix provides the complete experimental setup, including benchmark and model configurations, prompts and training details, system protocol, and additional implementation details. Anonymous source code, evaluation scripts, and configurations are provided in the abstract to facilitate reproduction of the reported results.

AI USE STATEMENT

We used AI tools to assist with running experiments and to polish the language of the manuscript for clarity and readability. The authors take full responsibility for the experimental results, analyses, and the final content of the paper.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 3119–3137, 2024.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

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

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988, 2019.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

Pengfei Du. Memory for autonomous llm agents: Mechanisms, evaluation, and emerging frontiers. *arXiv preprint arXiv:2603.07670*, 2026.

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

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, 2026. URL https://arxiv.org/abs/2510.18866.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024a.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. LightRAG: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779*, 2024b.

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*, 2024.

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

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2025.

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

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.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, 2024.

Yuyang Li, Zihe Yan, and Tobias Käfer. RASER: Recoverability-aware selective escalation router for multi-hop question answering. *arXiv preprint arXiv:2606.02488*, 2026.

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

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024b.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026.

Jaewan Park, Solbee Cho, and Jay-Yoon Lee. Stop-RAG: Value-based retrieval control for iterative RAG. *arXiv preprint arXiv:2510.14337*, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Mohammad Tavakoli, Alireza Salemi, Carrie Ye, Mohamed Abdalla, Hamed Zamani, and J. Ross Mitchell. Beyond a million tokens: Benchmarking and enhancing long-term memory in LLMs. In *International Conference on Learning Representations*, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

Kaixiang Wang, Yidan Lin, Jiong Lou, Zhaojiacheng Zhou, Bunyod Suvonov, and Jie Li. E-mem: Multi-agent based episodic context reconstruction for llm agent memory, 2026. URL https://arxiv.org/abs/2601.21714.

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.

B. Y. Yan, Chaofan Li, Hongjin Qian, Shuqi Lu, and Zheng Liu. General agentic memory via deep research. *arXiv preprint arXiv:2511.18423*, 2025.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024.

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

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, 2020.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. G-Memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*, 2025a.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞bench: Extending long context evaluation beyond 100k tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025b.

Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, Yuandong Tian, and Jishen Zhao. AMA-Bench: Evaluating long-horizon memory for agentic applications. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

A Limitations and Future Work

Router-Mem is currently trained with a finite evidence-supervision corpus that primarily covers general-purpose and everyday interaction scenarios. Expanding this supervision to specialized domains, such as medicine, law, finance, and scientific research, could further strengthen domain transfer and improve calibration under field-specific reasoning protocols. A second opportunity concerns the dependence between routing and retrieval. The router assesses the evidence exposed by the shared retrieval prefix; therefore, highly repetitive memories, ambiguous semantics, or weakly separated events may reduce retrieval precision and indirectly affect termination decisions. These cases motivate tighter coupling between retrieval confidence, evidence diversity, and routing, enabling the system to distinguish genuinely sufficient evidence from redundant or semantically uncertain retrieval results.

Future work will extend Router-Mem along both directions. We plan to construct domain-specific evidence-supervision pipelines and investigate lightweight adaptation strategies that preserve general routing ability while improving specialized transfer. We will also develop retrieval-aware routing objectives that incorporate confidence, redundancy, temporal consistency, and evidence coverage. Jointly optimizing the retriever and router, together with ambiguity detection and fallback expansion, may further improve robustness when memories contain repeated, noisy, or weakly distinguishable events, while retaining the framework's progressive and resource-efficient execution behavior.

Token Consumption. Table 6 compares end-to-end token usage across memory systems on AMA-Bench. At $\tau = 0.5$, Router-Mem achieves the highest score of 55.17% with 31.8K tokens per query. Relative to GAM, it cuts token consumption by 76.6% and end-to-end latency by 54.2%, while improving the score by 14.96 points. It uses 55.7% fewer tokens than Long Context and 72.1% fewer than Mem0, with score gains of 8.09 and 19.27 points, respectively. Relative to LightMem, Router-Mem uses 87.3% fewer tokens and gains 19.83 score points, although LightMem has lower end-to-end latency. Thus, Router-Mem's efficiency does not come from aggressive memory compression; it preserves substantially stronger answer quality while avoiding excessive context processing and generation overhead in deep or full-history memory systems.

Table 6: E2E token consumption on AMA-Bench. Input and output tokens include all online memory-processing and answer-generation calls.

Method	Score (%)	T_{e2e} (s)	Input Tokens	Output Tokens	Total Tokens
Router-Mem ($\tau = 0.5$)	**55.17**	11.57	**30,916.50**	**894.14**	**31,810.64**
GAM	40.21	25.289	128,417.35	7,578.30	135,995.65
LightMem	35.34	**2.30**	133,840.12	116,237.22	250,077.34
Long Context	47.08	5.18	71,633.30	201.74	71,835.04
Mem0	35.90	3.50	110,435.16	3,483.18	113,918.34

B System Execution Details

This section describes the complete inference procedure of Router-Mem. The system organizes long-term memory access as a progressive execution process. Every query first executes a shared retrieval prefix. The retrieved context is then evaluated by a lightweight evidence-sufficiency router. The system either returns an answer immediately or resumes the remaining memory-processing stages from the same retrieval state. Algorithm 2 summarizes the complete online procedure.

B.1 Aligned Memory Construction

Let the formulation

$$M = (z_1, \ldots, z_T) \tag{19}$$

denote an ordered long-term memory, where each step z_t may contain a dialogue turn, tool invocation, environment observation, intermediate decision, or execution feedback. Router-Mem constructs two aligned views of the same memory.

The *retrieval view* encodes individual memory steps into a dense index. Each indexed entry stores its original `step_id` and the identifier of the memory block containing that step. The *continuation view*

groups temporally adjacent steps into a sequence of partially overlapping blocks:

$$\mathcal{B} = (b_1, \ldots, b_J). \tag{20}$$

The alignment between the two views allows every retrieval hit to be mapped directly to one or more source blocks. Therefore, the retrieval result can serve both as an initial answering context and as a localization signal for deeper memory processing.

Memory construction is performed once for each episode or conversation. The dense index and block representation are then reused across all queries associated with the same memory.

B.2 Shared Retrieval Prefix

Given a query q, the trained retriever performs lightweight dense retrieval over the step-level index:

$$(C_f, H_f) = R_f(q, M), \tag{21}$$

where H_f contains the ranked retrieval hits and their metadata, and C_f is the rendered context formed from the retrieved memory steps.

The retrieved steps are kept with their original identifiers and timestamps. This information is used to restore temporal order for answer generation and to locate the corresponding continuation blocks when deeper processing is required. The shared retrieval prefix is executed for every query and is reused by both subsequent execution paths.

B.3 Evidence-Sufficiency Decision

The trained router reads the query and retrieved context (q, C_f) and produces a next-token distribution. Let V_{yes} and V_{no} denote the token groups corresponding to the normalized decisions `yes` and `no`. We aggregate their probability masses:

$$p_{\text{yes}} = \sum_{v \in V_{\text{yes}}} P_\theta(v \mid q, C_f), \qquad p_{\text{no}} = \sum_{v \in V_{\text{no}}} P_\theta(v \mid q, C_f), \tag{22}$$

and compute the evidence-sufficiency score

$$s_\theta(q, C_f) = \frac{p_{\text{yes}}}{p_{\text{yes}} + p_{\text{no}}}. \tag{23}$$

Given a threshold τ, Router-Mem follows the policy

$$\pi_\tau(q, C_f) = \begin{cases} \text{terminate}, & s_\theta(q, C_f) \geq \tau, \\ \text{continue}, & s_\theta(q, C_f) < \tau. \end{cases} \tag{24}$$

A larger threshold produces a more conservative policy and sends more queries to the continuation stage. A smaller threshold increases the early-termination rate. The threshold is fixed during each evaluation run and provides an explicit control over the empirical quality–latency trade-off.

If the retrieved context cannot be represented within the router input budget, the system conservatively selects continuation. This prevents an early-termination decision from being made on a partially truncated evidence context.

B.4 Optimistic Early Return

When the termination condition is satisfied, Router-Mem restores the retrieved steps to their original temporal order and passes them to the shared answer model:

$$\hat{a}_f = A\big(q, \text{Order}(C_f)\big). \tag{25}$$

This path is an early return from the complete memory execution plan. No block expansion, memory-agent processing, or evidence aggregation is performed. The system therefore pays only for retrieval, routing, and final answer generation.

B.5 Execution Continuation

When the retrieved evidence is predicted to be insufficient, the system continues from the shared-prefix state rather than restarting memory search. Let

$$\mathcal{B}_f = \mathrm{MapToBlocks}(H_f) \tag{26}$$

denote the source blocks associated with the retrieval hits. These blocks act as anchors for the continuation stage.

Router-Mem expands the anchor set into a bounded collection of processing blocks:

$$\mathcal{B}_s = \mathrm{SelectBlocks}\left(\mathcal{B}_f, M; K_b\right), \tag{27}$$

where K_b is the block-processing budget. The selection procedure first preserves blocks containing retrieval hits. It then includes temporally neighboring blocks to recover local action–observation, decision–feedback, and cross-step dependencies. When budget remains, additional blocks may be selected to improve coverage beyond the immediate retrieval neighborhoods.

Each selected block is processed independently:

$$u_b = \mathrm{MemAgent}(q, b), \qquad b \in \mathcal{B}_s. \tag{28}$$

The memory agent extracts query-relevant evidence and produces a block-local analysis. Because block processing is independent, the memory-agent calls can be executed in parallel.

An aggregator combines the block-level outputs:

$$C_s = \mathrm{Aggregate}\left(\{u_b : b \in \mathcal{B}_s\}\right). \tag{29}$$

The aggregator resolves duplicated evidence, preserves temporal order, and integrates complementary information across blocks. The same answer model used by the early-return path then generates

$$\hat{a}_s = A(q, C_s). \tag{30}$$

The fast and slow paths are therefore coupled through the shared retrieval prefix. The continuation stage preserves the retrieved evidence and reuses its block-level localization signals, avoiding a second memory search from scratch.

B.6 Runtime Cost Decomposition

For a query that terminates after the shared prefix, the online latency can be decomposed as

$$T_{\mathrm{fast}} = T_{\mathrm{ret}} + T_{\mathrm{router}} + T_{\mathrm{answer}}(C_f). \tag{31}$$

For a query that enters the continuation stage, the latency is

$$\begin{aligned} T_{\mathrm{slow}} = {} & T_{\mathrm{ret}} + T_{\mathrm{router}} + T_{\mathrm{select}} \\ & + T_{\mathrm{agents}} + T_{\mathrm{agg}} + T_{\mathrm{answer}}(C_s). \end{aligned} \tag{32}$$

When memory-agent calls are executed in parallel, T_{agents} is primarily determined by the slowest selected block rather than by the sum of all block-processing times. The expected online cost of Router-Mem therefore depends on the early-termination rate induced by τ and the cost of the continuation stage.

C Evidence-Sufficiency Data and Training Details

This section provides the complete data construction and training configuration for the retriever and sufficiency router in Router-Mem. The supervision is constructed from long-horizon agent trajectories that are source-disjoint from the downstream evaluation benchmarks. The training data preserve similar interaction structures, including tool use, environment feedback, multi-step decisions, and temporally dispersed evidence.

Algorithm 2 Progressive Memory Execution in Router-Mem

Require: Query q, memory M, threshold τ, block budget K_b
Ensure: Answer $\hat{a}$
$(C_f, H_f) \leftarrow R_f(q, M)$ ▷ Shared retrieval prefix
if C_f fits the router input budget **then**
  $s \leftarrow s_\theta(q, C_f)$
  if $s \geq \tau$ **then**
    $\hat{a} \leftarrow A\big(q, \mathrm{Order}(C_f)\big)$
    return $\hat{a}$ ▷ Optimistic early return
  end if
end if
$\mathcal{B}_f \leftarrow \mathrm{MapToBlocks}(H_f)$
$\mathcal{B}_s \leftarrow \mathrm{SelectBlocks}(\mathcal{B}_f, M; K_b)$
for all $b \in \mathcal{B}_s$ **in parallel do**
  $u_b \leftarrow \mathrm{MemAgent}(q, b)$
end for
$C_s \leftarrow \mathrm{Aggregate}\left(\{u_b : b \in \mathcal{B}_s\}\right)$
$\hat{a} \leftarrow A(q, C_s)$
return $\hat{a}$

Table 7: Statistics of the training corpora.

Dataset	Train	Validation	Total
Router supervision	124,344	13,590	137,934
Retriever supervision	60,690	6,584	67,274

C.1 Training Data Overview

We first normalize each agent trajectory into an ordered sequence of memory steps. A step may correspond to a dialogue turn, an agent action, a tool or API invocation, an environment observation, an execution result, or an error message. Each step is assigned a unique identifier and rendered as an explicit indexed block. This representation allows the data pipeline to associate each question with a verifiable subset of supporting memory steps.

The data construction pipeline produces two training corpora. The router corpus contains binary evidence-sufficiency examples, while the retriever corpus contains query, positive-evidence, and hard-negative groups. Their final sizes are summarized in Table 7.

The intermediate QA corpus contains 15,102 verified question–answer pairs derived from 3,683 trajectories. Each pair is associated with one to six supporting steps. Specifically, 8,880 pairs use one evidence step, 4,429 use two steps, and 1,793 require three or more steps. The training sources cover conversations, web navigation, tool use, software engineering, Text-to-SQL, and games.

C.2 QA–Evidence Construction and Verification

For each trajectory, a generator proposes a question q_i, a reference answer a_i, and a set of supporting step identifiers E_i. We use a mixture of three generators: DeepSeek-V4-Flash, Grok-4.3, and GPT-5.1, with sampling weights of 0.55, 0.35, and 0.10, respectively. Candidates are filtered using deterministic validity and near-duplication checks. The selected evidence must contain between one and six valid step identifiers and must not exceed 5,000 approximate tokens. We also reject candidates with excessive question–answer lexical overlap or high similarity to previously accepted questions from the same trajectory.

The remaining candidates are evaluated by a separate verifier. The verifier checks whether the selected evidence jointly supports the reference answer, whether the answer directly addresses the question, and whether any evidence step is clearly redundant. Only verified question–answer–evidence tuples

$$(q_i, a_i, E_i)$$

are passed to the second construction stage.

C.3 Evidence Intervention

We construct router supervision by intervening on the evidence available for the same verified question. A sufficient context retains all required evidence and mixes it with randomly sampled distractor steps:

$$C_i^+ = E_i \cup D_i, \tag{33}$$

where D_i denotes distractors sampled from the same trajectory. Therefore, a positive sample does not require every context step to be relevant; it only requires the complete answer-supporting evidence to be present.

An insufficient context removes part or all of the required evidence and fills the remaining context with random or semantically similar hard distractors:

$$C_i^- = \left(E_i \setminus E_i^{\text{drop}}\right) \cup D_i^{\text{hard}}. \tag{34}$$

Hard distractors are retrieved using the base Qwen3-Embedding-0.6B model. They may share entities, tools, or nearby events with the question, but do not preserve the complete supporting evidence. This construction prevents the router from reducing evidence sufficiency to topical relevance.

The target number of context steps is sampled from $\{6, 7, 8, 9, 10\}$ with probabilities $\{0.12, 0.22, 0.32, 0.22, 0.08\}$. The question and context are limited to 20,000 approximate tokens. If a sampled context exceeds this budget, we resample its distractors and, when necessary, reduce the target context size while retaining all required positive evidence.

Removing annotated evidence does not guarantee that the remaining context is unanswerable because redundant or alternative support may remain. We therefore apply false-negative filtering. Multiple answer models attempt to answer the question using only the candidate negative context. A separate judge compares each candidate answer with the reference answer. A negative context is discarded if any answer is judged correct. This procedure reduces incorrectly labeled negative examples, although it does not provide a formal guarantee of universal unanswerability.

For every retained router sample, a reasoning-capable model generates a label-aligned evidence-sufficiency rationale r_i. A sample is accepted only if its final decision exactly matches the expected `yes` or `no` label, its rationale is non-empty and no longer than 8,000 tokens, and the complete teacher input remains below 28,000 tokens. The final router records are represented as

$$(q_i, C_i, y_i, r_i), \qquad y_i \in \{\texttt{yes}, \texttt{no}\}.$$

The retriever records contain the query, one or more evidence-complete positive contexts, and an evidence-incomplete hard negative.

All data are split by trajectory identifier rather than by individual QA pair or augmented context. We shuffle unique trajectory identifiers with seed 42 and assign 10% to validation. Consequently, questions, neighboring steps, and context variants derived from the same trajectory cannot appear in both training and validation sets.

C.4 Embedding Retriever Training

The shared retriever is initialized from Qwen3-Embedding-0.6B. It uses a SentenceTransformer-style architecture consisting of a Transformer encoder, last-token pooling, and L2 normalization. The resulting embedding dimension is 1,024. For a query q_i, a positive set P_i, and negative set N_i, we optimize a multi-positive InfoNCE objective:

$$\mathcal{L}_{\text{emb}}^i = -\frac{1}{|P_i|} \sum_{p \in P_i} \log \frac{\exp\left(\text{sim}(q_i, p)/T_{\text{emb}}\right)}{\sum_{c \in P_i \cup N_i} \exp\left(\text{sim}(q_i, c)/T_{\text{emb}}\right)}, \tag{35}$$

where $\text{sim}(\cdot, \cdot)$ is the inner product between L2-normalized embeddings and $T_{\text{emb}} = 0.05$.

Table 8: Training configuration of the evidence-aware retriever.

Category	Parameter	Value
Backbone	Model	Qwen3-Embedding-0.6B
Data	Train / validation records	60,690 / 6,584
Randomness	Seed	42
Hardware	Training devices	One NVIDIA RTX 4090
Parallelism	World size	1
Optimization	Optimizer	AdamW
Optimization	Learning rate / weight decay	2.0×10^{-5} / 1.0×10^{-5}
Batching	Per-device batch / gradient accumulation	1 / 64; effective batch size 64 query groups
Training length	Epochs / final step	1 / 949
Schedule	Linear warmup	100 optimizer steps
Precision	Numeric format	bf16
Memory	Gradient checkpointing / use cache	enabled / disabled
Input	Maximum sequence length	32,768 tokens
Representation	Pooling / dimension / normalization	last token / 1,024 / L2 normalization
Loss	Temperature	0.05
Gradient	Maximum gradient norm	1.0
Evaluation	Evaluation and saving interval	Every 60 optimizer steps
Metrics	Retrieval cutoffs	HitAny and Recall at $k \in \{1, 3, 5\}$; HitAll@5 and MRR@5

The multi-positive objective allows different valid evidence views of the same question to remain positive. The hard negatives remain semantically related to the query but omit required evidence, encouraging the retriever to prioritize answer-supporting information rather than topic similarity alone.

Table 8 reports the complete retriever configuration. The run uses one RTX 4090 GPU and finishes at optimizer step 949 after one epoch.

On the 6,584-record validation set, the final retriever achieves HitAny@1 of 0.8305, HitAny@3 of 0.9646, and HitAny@5 of 0.9916. Its Recall@1, Recall@3, and Recall@5 are 0.6565, 0.8693, and 0.9442, respectively. HitAll@5 is 0.8855 and MRR@5 is 0.8976.

C.5 Rationale-Distilled Router Training

The router is initialized from Qwen3-1.7B. We denote the trainable student by S_θ and the frozen teacher by T_ϕ. Both use the same backbone architecture, but receive different inputs. The student reads only the question and context:

$$x_S^i = \mathrm{Prompt}(q_i, C_i), \tag{36}$$

whereas the teacher additionally receives the stored rationale:

$$x_T^i = \left[x_S^i; \texttt{<think>}r_i\texttt{</think>}\right]. \tag{37}$$

The teacher is used only during training and its parameters ϕ are never updated.

Grouped-token classification. The student retains the causal language-model output interface. We do not attach a separate classification head. Let V_{yes} and V_{no} contain every vocabulary token whose decoded text, after whitespace removal and lowercasing, equals `yes` or `no`. For the target label y_i, we optimize

$$\mathcal{L}_{\mathrm{cls}}^i = -\log \sum_{v \in V_{y_i}} P_\theta(v \mid x_S^i). \tag{38}$$

This grouped objective handles capitalization and leading-space tokenization variants while preserving the original language-model vocabulary.

For the tokenizer used in our experiments, the `yes` token group is

$$V_{\mathrm{yes}} = \{7414, 9454, 9693, 9834, 14004, 14080\},$$

Table 9: Training configuration of the evidence-sufficiency router.

Category	Parameter	Value
Backbone	Model	Qwen3-1.7B
Data	Train / validation records	124,344 / 13,590
Randomness	Seed	42
Hardware	Training devices	Two NVIDIA RTX 4090 GPUs
Parallelism	Distributed training	DDP with world size 2
Optimization	Optimizer	AdamW
Optimization	Learning rate / weight decay	1.0×10^{-4} / 0
Batching	Per-device batch / gradient accumulation	1 / 64; nominal global batch size 128
Training length	Epochs / selected checkpoint	3 / step 1,560
LoRA	Rank / alpha / dropout	16 / 32 / 0.0
LoRA	Adapted layers	All decoder layers
Precision	Numeric format	bf16
Memory	Gradient checkpointing	enabled
Input	Maximum sequence length	28,000 tokens
Loss	λ_{rep}	0.3
Loss scaling	$s_{\min}$ / $s_{\max}$ / ϵ	0.1 / 10.0 / 10^{-8}
Warmup	CE-only steps / warmup ratio	50 / 0.2
Gradient	Maximum gradient norm	1.0
Evaluation	Evaluation and saving interval	Every 60 optimizer steps on the full validation set
Latency check	Samples / warmup runs	100 / 3

and the `no` token group is

$$V_{\text{no}} = \{902, 2152, 2308, 2753, 5664, 8996, 72104\}.$$

Representation distillation. A binary label provides limited supervision about why a context is sufficient or insufficient. We therefore transfer the teacher's rationale-conditioned representation to the student. Let L be the number of decoder layers and

$$l^* = \left\lfloor \frac{5L}{6} \right\rfloor.$$

We extract the hidden state of the last non-padding token at layer l^* from both models, denoted by h_S^i and h_T^i. The representation loss is

$$\mathcal{L}_{\text{rep}}^i = 1 - \cos\left(\text{sg}(h_T^i), h_S^i\right), \tag{39}$$

where sg stops gradients through the frozen teacher.

The combined router objective is

$$\mathcal{L}_{\text{router}} = \mathcal{L}_{\text{cls}} + \lambda_{\text{rep}} s_t \mathcal{L}_{\text{rep}}, \tag{40}$$

where $\lambda_{\text{rep}} = 0.3$. Training begins with 50 classification-only warmup steps. After warmup, the scale s_t is estimated from the exponential moving magnitudes of the classification and representation losses and clipped to $[0.1, 10.0]$. This calibration prevents either objective from dominating solely because of its numeric scale.

We fine-tune all decoder layers using LoRA. The target modules are `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`. The final LoRA weights are merged into the backbone to produce a directly loadable router checkpoint.

The selected router checkpoint is obtained at step 1,560 after consuming 199,680 training samples and approximately 2.50 billion training tokens. On the 13,590-example validation set, it achieves an accuracy of 0.9199, precision of 0.9150, recall of 0.9609, and F1 of 0.9374. The grouped `yes`/`no` probability mass is 0.9995, and no invalid top-1 or top-3 decisions are observed.

Table 10: Hardware and runtime environment.

Component	Configuration
GPU	NVIDIA RTX 4090, 24,564 MiB
Available RTX 4090 GPUs	4
CPU	2× Intel Xeon Platinum 8352V
CPU cores	72 physical / 144 logical
System memory	503 GiB
Operating system	Ubuntu 24.04.1 LTS
Linux kernel	6.17.0-35-generic
NVIDIA driver	595.58.03
CUDA	13.2

C.6 Inference Score and Thresholds

At inference time, only the trained student S_θ is retained. The rationale and frozen teacher are removed. Given the first-token log-probabilities, we aggregate the two semantic token groups:

$$p_{\text{yes}} = \sum_{v \in V_{\text{yes}}} P_\theta(v \mid q, C_f), \qquad p_{\text{no}} = \sum_{v \in V_{\text{no}}} P_\theta(v \mid q, C_f), \tag{41}$$

and compute

$$s_\theta(q, C_f) = \frac{p_{\text{yes}}}{p_{\text{yes}} + p_{\text{no}}}. \tag{42}$$

The threshold τ is not optimized jointly with the model. We evaluate the operating points

$$\tau \in \{0.1, 0.3, 0.5, 0.7, 0.9\}.$$

Unless otherwise specified, the main experiments use $\tau = 0.5$ as the representative operating point because it provides a favorable quality–latency trade-off. The complete threshold sweep is reported to show how the controller moves between aggressive early termination and conservative full memory processing.

D Experimental Implementation Details

This section provides the implementation and evaluation details used to reproduce the experiments in the main paper. We first describe the hardware and common evaluation protocol, and then report the benchmark-specific configurations, baseline controls, latency definitions, and run-management procedures.

D.1 Hardware and Runtime Environment

All system-level evaluations are conducted on NVIDIA GeForce RTX 4090 GPUs. The evaluation server contains four RTX 4090 GPUs, each with 24,564 MiB of memory. The machine is equipped with two Intel Xeon Platinum 8352V processors, providing 72 physical cores and 144 logical threads, and 503 GiB of host memory. It runs Ubuntu 24.04.1 LTS with Linux kernel 6.17.0-35-generic, NVIDIA driver 595.58.03, and CUDA 13.2. Although the server also contains other GPU models, we explicitly select RTX 4090 devices through `CUDA_VISIBLE_DEVICES` for all reported system experiments.

The evidence-sufficiency router and embedding retriever are served locally during evaluation. External language-model calls are issued through their corresponding API endpoints. Unless otherwise specified, all generation models run with reasoning or thinking mode disabled to reduce uncontrolled variation in generation length and latency.

D.2 Common Evaluation Protocol

We use DeepSeek-V4-Flash as the primary answer model, slow-path memory agent, and evidence aggregator. The maximum generation length is 4,096 tokens. The fast and slow paths use the same

final answer model; their only difference is the memory context supplied to that model. This control ensures that quality differences arise from memory processing rather than from different answer backbones.

For each benchmark, we retain its official scoring protocol and LLM judge. The reported *Score* is therefore the benchmark-level answer quality produced by the original evaluation pipeline, rather than a separately designed metric. Judge execution time is excluded from inference latency because judging is an offline evaluation operation and is not part of the deployed memory system.

The primary Router-Mem operating point uses $\tau = 0.5$. We additionally evaluate

$$\tau \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$$

to characterize the quality–latency frontier. All threshold variants share the same retriever, router checkpoint, answer model, benchmark split, and evaluation pipeline. Only the termination threshold changes.

D.3 Latency and Token Accounting

We report three complementary latency measurements.

- **Memory construction time** T_{build} is the time required to construct the memory representation for one episode or conversation. It includes preprocessing, embedding computation, index construction, and any method-specific memory-building operations.
- **Online memory time** T_{mem} measures query-time memory processing. For Router-Mem, this includes shared retrieval, router inference, and, when triggered, block expansion, memory-agent execution, and aggregation. It excludes final answer generation.
- **End-to-end time** T_{e2e} measures the complete online path from receiving the query to obtaining the final answer:
$$T_{\mathrm{e2e}} = T_{\mathrm{mem}} + T_{\mathrm{answer}}. \tag{43}$$

Unless otherwise stated, latency values are averaged per query. They are measured as observed wall-clock time under the deployed local and API configuration. Consequently, absolute values may be affected by API service load and network variation, while comparisons within the same run configuration remain directly comparable.

We separately record input and output tokens for memory processing and answer generation. End-to-end token consumption includes all online model calls made after the query arrives. Offline data-generation tokens and one-time training costs are not included in online token totals. Memory-construction tokens are reported separately when lifecycle cost is analyzed.

Early termination is defined as

$$\mathrm{ETR} = \frac{\#\{\text{queries answered after the shared prefix}\}}{\#\{\text{all evaluated queries}\}}. \tag{44}$$

A query is counted as early-terminated only when the system directly answers from the retrieved context without invoking the slow continuation.

D.4 Router-Mem Online Implementation

For each episode or conversation, Router-Mem constructs two aligned memory views. The retrieval view indexes individual memory steps and stores their source step and block identifiers. The continuation view groups temporally neighboring steps into memory blocks. This alignment allows each retrieval hit to identify the corresponding region used by the slow continuation.

Shared retrieval prefix. Given query q, the trained Qwen3 embedding retriever searches a local-disk Qdrant index. The fast prefix retrieves the top eight memory steps:

$$C_f, H_f = R_f(q, M), \tag{45}$$

where C_f is the rendered evidence context and H_f contains the retrieval metadata. We use an embedding batch size of two and a context truncation ratio of 0.8.

The Qwen3-1.7B router consumes (q, C_f) and outputs first-token log-probabilities. We aggregate the normalized `yes` and `no` token groups to obtain

$$s_\theta(q, C_f) = \frac{p_{\text{yes}}}{p_{\text{yes}} + p_{\text{no}}}. \tag{46}$$

If $s_\theta(q, C_f) \geq \tau$, the retrieved steps are restored to their original temporal order and passed directly to the answer model.

Slow continuation. When $s_\theta(q, C_f) < \tau$, Router-Mem retains the retrieval hits and maps them to their source blocks. These blocks serve as anchors for broader memory expansion. Selected blocks are independently processed by memory agents and then consolidated by an aggregator:

$$C_s = \text{Aggregate}\left(\{\text{MemAgent}(q, b) : b \in \mathcal{B}_s\}\right). \tag{47}$$

The same answer model then generates $A(q, C_s)$. Thus, the continuation reuses the shared-prefix computation instead of restarting memory search from scratch.

The memory agents and aggregator use DeepSeek-V4-Flash with thinking disabled and a maximum output length of 4,096 tokens. Independent block calls are executed concurrently when permitted by the runtime configuration.

D.5 AMA-Bench Configuration

We evaluate on the complete AMA-Bench `openend` split, which contains 208 episodes and 2,496 questions. Questions from the same episode are processed sequentially:

$$\texttt{max_concurrency_questions_per_episode} = 1.$$

This preserves the temporal order of memory construction and prevents later questions from observing a memory state that should not yet exist.

The primary answer model is DeepSeek-V4-Flash with thinking disabled and a 4,096-token output limit. AMA-Bench scores are computed by the benchmark's original LLM-judge pipeline. The judge uses Qwen3-32B with thinking disabled and a maximum output length of 1,024 tokens. Up to 32 judge workers are used for the complete threshold sweep. Judge parallelism changes offline evaluation throughput but does not alter the per-episode query order or the online Router-Mem execution.

For Router-Mem, the fast prefix uses

$$k_{\text{fast}} = 8, \qquad \texttt{truncate_ratio} = 0.8.$$

The slow continuation uses at most ten memory blocks. Each block is limited to 8,000 tokens, and the aggregator receives at most 16,000 tokens of consolidated memory. Block completion uses a neighbor-to- uniform fill ratio of 0.8 to 0.2, prioritizing temporal neighborhoods around retrieval anchors while retaining limited global coverage. The aggregator is enabled for all slow-path runs.

The principal AMA-Bench settings are summarized in Table 11.

D.6 BEAM Configuration

We use the 1M-token split of BEAM, containing 35 conversations and 700 questions. The benchmark covers ten question categories with 70 questions per category. We preserve the official final-report evaluation path, including rubric-level judge aggregation and the upstream event-ordering procedure.

The BEAM sweep includes Fast-only, Slow-only, and Router-Mem with thresholds

$$\{0.1, 0.3, 0.5, 0.7, 0.9\}.$$

Table 11: Benchmark-specific implementation settings.

Configuration	AMA-Bench	BEAM
Evaluation split	`openend`	1M-token split
Episodes / conversations	208	35
Questions	2,496	700
Question categories	benchmark-defined	10
Questions per category	–	70
Fast retrieval top-k	8	8
General retrieval budget	10	15
Slow block budget	10	15
Embedding batch size	2	2
Router thresholds	$\{0.1, 0.3, 0.5, 0.7, 0.9\}$	
Primary answer model	DeepSeek-V4-Flash	
Thinking mode	disabled	
Maximum answer output	4,096 tokens	

RAG and Long Context are run as independent baseline configurations. All reported Router-Mem points are derived from complete 35-conversation, 700-question summaries. We do not interpolate or synthesize missing threshold results.

Retrieval-based methods use a total retrieval budget of 15 on BEAM, while the shared fast prefix remains at $k_{\text{fast}} = 8$ to fit the router input budget. GAM performs three iterative retrieval rounds under the same total retrieval budget of 15. The embedding batch size is fixed to two for all methods that use batched embedding computation.

D.7 Baseline Implementations and Fairness Controls

We compare Router-Mem with RAG, LightMem, Mem0, Long Context, and GAM, as well as the two fixed Router-Mem endpoints.

- **RAG** performs one-shot dense retrieval followed by answer generation.
- **LightMem** uses its compact memory-construction and retrieval pipeline before answer generation.
- **Mem0** extracts and consolidates persistent memory entries and retrieves the entries associated with each query.
- **Long Context** directly supplies the available history to the answer model without an external retrieval stage.
- **GAM** performs three rounds of query-time retrieval and memory reconstruction. Its total retrieval budget is constrained to the same benchmark-level budget used by other retrieval methods.
- **Fast-only** always answers after the shared retrieval prefix and therefore has an early-termination rate of 100%.
- **Slow-only** always executes the complete memory continuation and has an early-termination rate of 0%.

All methods use the same benchmark split, final answer model, and official evaluation protocol. Retrieval-based baselines use the same embedding batch size. On AMA-Bench, their total retrieval budget is ten; on BEAM, it is fifteen. The Router-Mem fast prefix uses eight retrieved steps because the router must jointly encode the query and retrieved context within its input budget.

E Complete Prompt Templates

This appendix reproduces verbatim the prompts used for QA–evidence construction, negative-context filtering, router training and inference, and online memory processing. Braced expressions such as {`question`} denote fields instantiated at runtime. We retain the original English wording to preserve the exact model interfaces.

Table 12: Memory access configurations used for baseline comparison.

Method	Memory access	AMA budget	BEAM budget
RAG	One-shot dense retrieval	$k = 10$	$k = 15$
LightMem	Compact memory construction and retrieval	$k = 10$	$k = 15$
Mem0	Extracted persistent memory and retrieval	$k = 10$	$k = 15$
Long Context	Complete available history	Full context	Full context
GAM	Three iterative retrieval rounds	Total $k = 10$	Total $k = 15$
Fast-only	Shared retrieval prefix	$k = 8$	$k = 8$
Slow-only	Complete block-level continuation	10 blocks	15 blocks
Router-Mem	Prefix plus thresholded continuation	$k = 8$, up to 10 blocks	$k = 8$, up to 15 blocks

E.1 QA–Evidence Construction

QA Generator: System Prompt

```
You generate evidence-grounded QA pairs from normalized agent
    trajectory steps.

Rules:
- Use English only.
- The question and answer must be fully supported by the selected
    evidence steps.
- Do not rely on outside knowledge, assumptions, subjective evaluation,
     counterfactuals, or guesses about model intent.
- Prefer questions that test whether retrieved context contains
    sufficient evidence.
- Select the minimal evidence set needed to answer the question. Use 1
    to 6 steps.
- Evidence ids must be the numeric step ids shown in the input.
- Do not repeat questions already accepted for this trace.

Return one JSON object only:
{
  "question": "...",
  "answer": "...",
  "evidence_step_ids": [1, 2],
  "why_each_evidence_is_needed": {
    "1": "...",
    "2": "..."
  }
}
```

QA Generator: User Template

```
Domain: {domain}
Source: {source}
Already accepted questions for this trace:
{one question per line, or "- None yet"}

Normalized steps:
{indexed <step> blocks for the entire trace}
```

QA Verifier: System Prompt

```
You verify whether a proposed QA pair is supported by selected evidence.

Check:
- The selected evidence steps, together, clearly support the gold
    answer.
- The gold answer directly answers the question.
- Every selected evidence step is useful. Do not be overly strict: mark
     evidence redundant only when it is clearly unnecessary.
- Use only the selected evidence. Do not use outside knowledge.

Return one JSON object only:
{
  "supported": true,
  "answer_equivalent": true,
  "redundant_evidence_ids": [],
  "brief_reason": "..."
}
```

QA Verifier: User Template

```
Question:
{question}

Gold Answer:
{answer}

Selected Evidence Steps:
<evidence id="{step_id}">
{step text}
</evidence>
...
```

E.2 Negative-Context Filtering

Negative Answerer: System Prompt

```
Answer the question using only the supplied context.

If the context does not contain enough evidence, return an empty answer
     string.

Return one JSON object only:
{
  "answer": "..."
}
```

Negative Answerer: User Template

```
Question:
{question}
```

```
Context:
{indexed candidate context}
```

Answer Judge: System Prompt

```
Judge whether the candidate answer is semantically correct for the
    question compared with the gold answer.

Ignore wording differences. Mark correct only if the candidate answer
    provides the same required information as the gold answer.

Return one JSON object only:
{
  "correct": false,
  "brief_reason": "..."
}
```

Answer Judge: User Template

```
Question:
{question}

Gold Answer:
{gold answer}

Candidate Answer:
{answer generated from the negative context}
```

E.3 Router Training and Inference

The trainable student receives the router system and user prompts. The frozen teacher receives the same input followed by the stored, label-aligned rationale extension. At inference time, the teacher and rationale are removed, and only the trained student is retained.

Evidence-Sufficiency Router: System Prompt

```
Decide whether the retrieved context is sufficient to answer the
    question.

Do not use outside knowledge. Do not require exact wording, but require
     clear evidence.

In the final assistant content, output exactly one token (yes or no):
- yes: the context contains enough evidence to answer the question.
- no: the context does not contain enough evidence to answer the
    question.
```

Evidence-Sufficiency Router: User Template

```
Question:
{question}

Retrieved Context:
```

```
<step id="1">
{retrieved step 1}
</step>
...

Can the retrieved context sufficiently answer the question? Return only
     yes or no.
```

Frozen Teacher: Rationale Extension

```
<think>
{stored rationale}
</think>
```