MEMPILOT: ORCHESTRATING ON-DEMAND MULTIMODAL MEMORY CURATION FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory has become integral to the LLM agent ecosystem, supporting information retention and reuse across interactions. However, most existing agent memory systems construct memory in a query-agnostic manner, which can incur unnecessary preprocessing cost and discard details that later prove essential. Recent studies have begun shifting memory processing toward runtime adaptation, but typically specialize in particular operations or fixed processing schemes, leaving flexible control over performance, cost, and latency largely underexplored. To address this challenge, we present **MemPilot**, a flexible framework that orchestrates on-demand memory curation under different performance–cost–latency preferences. Specifically, we optimize a multi-step LLM policy via reinforcement learning to iteratively choose between retrieving from query-agnostic memory and delegating query-specific curation of raw multimodal history to heterogeneous LLMs and VLMs. The policy jointly controls evidence amount, curation instructions, model selection, and visual access, enabling fine-grained allocation of runtime computation. To optimize this policy under competing objectives, we adapt objective-wise advantage decoupling by separately estimating each objective's advantage before aggregation. Moreover, we introduce prefix-based marginal utility estimation for fine-grained credit assignment across multi-step rollouts. Experiments on five multimodal agent-memory benchmarks demonstrate favorable performance–cost–latency trade-offs across optimization preferences, with preference sweeps yielding broader frontiers than existing trade-off-aware baselines.

1 INTRODUCTION

As large language model (LLM) agents evolve toward persistent, long-horizon interaction, memory becomes increasingly central to preserving and reusing past experience (Hu et al., 2025; Huang et al., 2026). However, most existing agent memory systems construct memory from interaction histories in a query-agnostic manner, and relying solely on such pre-query construction can be wasteful and lossy, as it cannot reliably anticipate future query needs (Xu et al., 2025; Fang et al., 2025; Liu et al., 2026c). To mitigate this limitation, a growing line of work moves memory adaptation to runtime through query-time evidence distillation, on-demand memory construction, and adaptive memory strategies (Yan et al., 2025a; Zhang et al., 2026d). Despite this progress, flexible orchestration of runtime memory adaptation remains underexplored, particularly when adapting runtime computation to query needs and multimodal evidence while balancing performance, cost, and latency (Zhang et al., 2026b). This raises a key challenge: *how can runtime multimodal memory curation be adaptively orchestrated with controllable performance–cost–latency trade-offs?*

Unfortunately, addressing this challenge is non-trivial due to three key limitations of existing approaches to runtime memory adaptation. ❶ *Runtime memory control remains structurally rigid*: existing methods typically rely on hand-designed operations or fixed processing pipelines, with efficiency control hard-wired into predefined modules, budget tiers, or heuristics (Yin & Tang, 2026; Zhang et al., 2026b; Yu et al., 2026a), leaving few degrees of freedom for query-specific adaptation and making fine-grained trade-off optimization difficult. ❷ *Trade-off optimization remains incomplete*: existing approaches often target only a subset of efficiency objectives, such as token usage or monetary cost, while user-facing latency is rarely treated as an explicit optimization objective (Liu et al., 2026b; Zhang et al., 2026d). ❸ *Existing methods largely assume a fixed processing backend for runtime memory adaptation*, overlooking substantial variation in curation quality, cost,

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.

and latency across heterogeneous models and limiting adaptation to query difficulty and evidence modality, thereby constraining the achievable trade-off space (Yu et al., 2026a; Yan et al., 2025a). Collectively, these limitations call for a more flexible framework that can orchestrate runtime memory operations and processing backends according to both query needs and optimization preferences.

Building on these observations, in this work we introduce **MemPilot**, a flexible framework for orchestrating on-demand memory curation under different performance–cost–latency preferences. At a high level, MemPilot follows a simple design principle: retain query-agnostic memory, compatible with off-the-shelf agent memory systems, for efficient access, while preserving raw interaction history for query-specific runtime curation. Specifically, at inference time, a multi-step LLM policy constructs a query-adaptive orchestration trajectory by iteratively choosing between retrieving from the query-agnostic memory bank and delegating curation of the raw history to heterogeneous LLMs and vision-language models (VLMs), whose distinct capabilities naturally offer different quality, cost, and latency profiles. Rather than relying on hand-crafted pipelines or discrete budget tiers, the policy adaptively determines how much evidence to process, how it should be curated, which model to invoke, and whether visual evidences are needed, exposing fine-grained controls over runtime computation trade-offs. We optimize this policy with reinforcement learning under different performance, cost, and latency preferences. To better handle the resulting multi-objective optimization, we adapt objective-wise advantage decoupling by estimating each objective's advantage separately before weighted aggregation. We further introduce prefix-based marginal utility estimation, which estimates the marginal contribution of individual curation steps for fine-grained credit assignment.

Experiments on five multimodal agent-memory benchmarks show that MemPilot achieves favorable performance–cost–latency trade-offs across different optimization preferences. Fine-grained preference sweeps further reveal broader trade-off frontiers than existing trade-off-aware baselines, while behavioral analyses show that MemPilot adaptively reallocates runtime computation as optimization preferences change. **Our contributions are summarized as follows:**

- We introduce **MemPilot**, a systematic framework for preference-aware runtime memory orchestration that jointly controls memory access, query-specific curation, heterogeneous model delegation, and visual evidence usage. By unifying these decisions within a multi-step policy, MemPilot enables fine-grained performance–cost–latency control without relying on fixed processing pipelines, predefined budget tiers, or a fixed processing backend.
- To optimize the orchestration policy under competing objectives, we adapt objective-wise advantage decoupling and introduce prefix-based marginal utility estimation for fine-grained credit assignment in multi-step rollouts, enabling stable and effective optimization.
- Experiments on five multimodal agent-memory benchmarks demonstrate favorable performance–cost–latency frontiers across preferences, with analyses revealing shifts in model selection, visual access, and computation allocation as preferences change.

2 Related Work

2.1 Agent Memory and Runtime Adaptation

LLM agents increasingly rely on memory mechanisms to preserve and reuse information from long-term interaction histories (Zhong et al., 2024; Packer et al., 2023). Most existing text-centric systems construct memory before future queries are known, using either hand-designed mechanisms for extraction and organization (Chhikara et al., 2025; Xu et al., 2025; Kang et al., 2025; Fang et al., 2025; Liu et al., 2026c) or learned policies for memory construction, updating, and retrieval (Yan et al., 2025b; Wang et al., 2025; Zhang et al., 2026a; Yu et al., 2026b; Yue et al., 2026). Beyond text-centric memory, recent work has extended agent memory to multimodal interactions, maintaining structured representations over textual, visual, and other sensory experiences (Feng et al., 2026a; Wang & Chen, 2025; Long et al., 2026; Liu et al., 2025; 2026b). Despite increasingly adaptive and multimodal memory designs, they still construct memory in a largely query-agnostic manner.

A growing line of work instead shifts memory processing toward runtime adaptation. GAM (Yan et al., 2025a) performs just-in-time research over stored memory and raw history, while Budget-Mem (Zhang et al., 2026b) learns query-aware runtime extraction under different computation budgets. MemGen (Zhang et al., 2025a) and AdaMEM (Zhang et al., 2026d) dynamically generate or

refresh memory guidance according to evolving reasoning states, whereas MemRL (Zhang et al., 2026c) adapts the utility of stored experiences from interaction feedback. More recent work further explores query-time evidence distillation and deferred memory construction (Yin & Tang, 2026; Yu et al., 2026a). However, these approaches typically specialize in a particular form of runtime adaptation or follow a predefined processing pattern. MemPilot instead learns to orchestrate heterogeneous multimodal memory curation at runtime under controllable performance–cost–latency preferences.

2.2 LLM Routing

LLM routing aims to match different queries with suitable models based on their heterogeneous capabilities and costs. Early approaches route queries through model cascades or select models based on estimated query difficulty (Chen et al., 2023; Ding et al., 2024). More recent work improves routing through preference learning (Ong et al., 2025), contrastive objectives (Chen et al., 2024), and graph-based modeling (Feng et al., 2024), while further extending routing to multi-agent settings (Yue et al., 2025) and unified routing frameworks (Feng et al., 2026b). Moving beyond one-shot routing, several methods employ reinforcement learning to enable multi-step model selection and orchestration (Zhang et al., 2025b; Qian et al., 2025; Feng et al., 2026c). These advances highlight the potential of LLM routing as a general paradigm for adaptively allocating heterogeneous model computation according to task demands and resource constraints.

3 Methodology

In this section, we first formulate on-demand memory orchestration as a multi-step decision process (Section 3.1). We then instantiate the factorized curation and delegation decisions that enable query-adaptive runtime control (Section 3.2). Finally, we introduce resource-aware reward modeling for optimizing quality–cost–latency trade-offs with fine-grained credit assignment (Section 3.3).

3.1 On-Demand Memory Orchestration as a Multi-Step Decision Process

Dual-view memory representation. To support flexible runtime adaptation while retaining efficient access to query-agnostic memory, MemPilot maintains two complementary views of the interaction history: a query-agnostic memory bank $\mathcal{M}$ and the raw multimodal history $\mathcal{H}$. In this work, $\mathcal{M}$ can be initialized with memories constructed by any off-the-shelf agent memory system (Xu et al., 2025; Fang et al., 2025; Liu et al., 2026c) or a simpler memory construction scheme, keeping memory construction orthogonal to MemPilot's runtime orchestration and ensuring broad compatibility with existing memory systems. Meanwhile, $\mathcal{H}$ is segmented into fixed-length token chunks while preserving the original multimodal context within each chunk. These two views serve complementary roles: the former enables efficient memory access, whereas the latter retains fine-grained information that may be absent from the query-agnostic memory representation.

Sequential decision formulation. As shown in Figure 1, given a query q, MemPilot formulates on-demand memory curation as a multi-step decision process governed by an orchestrator policy π_θ. At step t, the orchestrator conditions on the query and previously acquired evidence, $s_t = (q, a_{<t}, o_{<t})$, and produces a structured memory action:

$$a_t = (z_t, \phi_t) \sim \pi_\theta(\cdot \mid s_t), \tag{1}$$

where z_t denotes the action type and ϕ_t denotes its associated action arguments (for simplicity, we show one action per step, and multiple actions may be issued and run in parallel). Specifically,

$$z_t \in \{\textbf{RETRIEVE}, \textbf{CURATE}\}. \tag{2}$$

Here, **RETRIEVE** directly accesses the query-agnostic memory bank $\mathcal{M}$, while **CURATE** performs query-specific runtime curation over the raw history $\mathcal{H}$. We abstract the resulting observation as:

$$o_t = \begin{cases} \mathcal{R}_{\mathcal{M}}(\phi_t), & z_t = \textbf{RETRIEVE}, \\ \mathcal{C}_{\mathcal{H}}(\phi_t), & z_t = \textbf{CURATE}, \end{cases} \tag{3}$$

where $\mathcal{R}_{\mathcal{M}}(\cdot)$ denotes semantic-similarity retrieval over the compressed memory bank, while $\mathcal{C}_{\mathcal{H}}(\cdot)$ denotes query-specific runtime curation over the raw history, including evidence retrieval and subsequent processing. In our instantiation, $\mathcal{C}_{\mathcal{H}}(\cdot)$ delegates the retrieved evidence to a heterogeneous

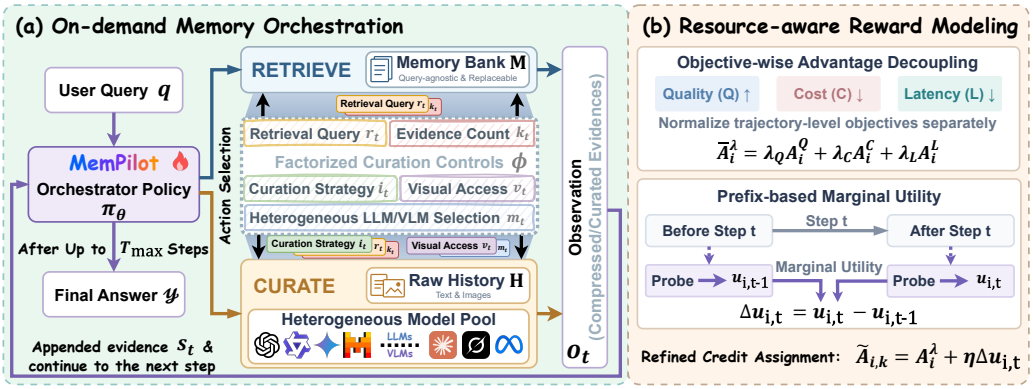


Figure 1: **Overview of MemPilot. (a)** The orchestrator iteratively retrieves compressed memories or delegates query-specific curation of raw multimodal histories to heterogeneous LLMs/VLMs. Factorized controls govern evidence retrieval, curation strategy, visual access, and model selection, with accumulated evidence guiding subsequent decisions and final answering. **(b)** Objective-wise advantage decoupling enables controllable trade-offs among quality, cost, and latency, while prefix-based answer probing estimates each step's marginal utility for fine-grained credit assignment.

pool of external LLMs and VLMs (elaborated in Section 3.2), whereas the orchestrator itself remains text-only and never directly processes raw multimodal inputs (e.g., image pixels). After a variable number of steps, the policy terminates memory curation and produces the final answer y.

This process yields a query-adaptive curation trajectory with at most $T_{\max}$ steps:

$$\tau = (q, a_1, o_1, \ldots, a_T, o_T, y), \quad T \le T_{\max}, \tag{4}$$

whose behavior can vary substantially across queries and optimization preferences. We characterize each trajectory by its answer quality $Q(\tau)$ and runtime resource consumption:

$$\mathbf{R}(\tau) = \big(C(\tau),\, L(\tau)\big), \tag{5}$$

where $C(\tau)$ denotes monetary cost and $L(\tau)$ denotes user-facing latency. Given an optimization preference $\boldsymbol{\lambda}$, our goal is to learn an orchestrator policy π_θ that flexibly balances answer quality $Q(\tau)$, monetary cost $C(\tau)$, and latency $L(\tau)$ rather than optimizing task performance alone. This formulation casts on-demand memory orchestration as a preference-conditioned sequential decision problem over answer quality and resources, laying the foundation for explicit trade-off optimization.

3.2 Factorized Memory Curation and Delegation

Factorized curation controls. We next instantiate the structured action arguments ϕ_t to expose fine-grained control over runtime memory curation. Rather than relying on a fixed processing pipeline or predefined budget tiers, MemPilot factorizes each action into several controllable dimensions, allowing the orchestrator to adapt *what evidence to retrieve*, *how much to retrieve*, *how it should be curated*, *which model should process it*, and *whether visual evidence is needed*. These decisions jointly provide flexible control over runtime computation and its resulting quality–cost–latency trade-offs.

We factorize the action arguments according to the two action types:

$$\phi_t = \begin{cases} (r_t, k_t), & z_t = \textsc{Retrieve}, \\ (r_t, k_t, i_t, m_t, v_t), & z_t = \textsc{Curate}, \end{cases} \tag{6}$$

where r_t is the retrieval query and k_t specifies the evidence count, allowing the orchestrator to adapt retrieval depth across steps rather than relying on a fixed top-k. For CURATE, i_t further specifies a query-specific curation instruction, m_t selects the processing backend, and v_t determines whether original visual evidence is included. Importantly, we decouple *what to retrieve* (r_t, e.g., "Lena's dress options") from *how to curate it* (i_t, e.g., "extract the two dress types she considered"), avoiding conflating two semantically distinct operations in a single textual field while keeping retrieval evidence-focused and curation task-directed.

Heterogeneous model delegation. Beyond deciding what evidence to retrieve and how to curate it, MemPilot also adapts which model performs the runtime curation, since heterogeneous models exhibit distinct capabilities, monetary costs, and latencies, making model choice a natural and effective control dimension for runtime trade-offs. Inspired by the LLM routing paradigm (Feng et al., 2024; Zhang et al., 2025b), MemPilot therefore delegates runtime curation to a configurable pool of LLMs and VLMs, allowing the orchestrator to adaptively select a model for each curation step under different optimization preferences. To support this routing decision, the orchestrator is given anonymized model IDs together with lightweight profiles describing key characteristics such as parameter scale and supported input modalities. The selected model m_t then processes the retrieved evidence according to i_t and returns a textual observation to the orchestrator. For vision-capable routes, v_t further determines whether raw images should be included in the curation request.

Together, these factorized decisions allow each query to induce its own curation trajectory over evidence depth, processing strategy, model capacity, and modality access, providing flexible trade-off control without requiring a rigid processing pipeline or predefined budget tiers.

3.3 Resource-Aware Reward Modeling

Having exposed multiple controls over runtime curation, we train the orchestrator to use them according to different optimization preferences. For a preference vector $\boldsymbol{\lambda} = (\lambda_Q, \lambda_C, \lambda_L)$ (with $\lambda_Q = 1$ throughout this work), we optimize the policy with Group Relative Policy Optimization (GRPO) (Shao et al., 2024) over answer quality, monetary cost, and latency.

Resource-aware GRPO. For each query, the policy π_θ samples a group of G curation trajectories $\{\tau_i\}_{i=1}^{G}$ and evaluates each trajectory with three objective-specific rewards:

$$r_i^Q = Q(\tau_i), \qquad r_i^C = -C(\tau_i), \qquad r_i^L = -L(\tau_i), \tag{7}$$

where $Q(\tau_i)$ is the task-specific answer reward (e.g., F1-Score). Monetary cost accounts for token-based inference costs incurred by both orchestrator generation and delegated model calls,

$$C(\tau_i) = C_\pi(\tau_i) + C_{\text{ext}}(\tau_i). \tag{8}$$

Latency similarly considers both sources of runtime delay. Since direct wall-clock measurements are sensitive to network and serving fluctuations, we use a proxy estimator $L(\tau_i) = \widehat{L}_\pi(\tau_i) + \widehat{L}_{\text{ext}}(\tau_i)$ to ensure training stability, with more details provided in Appendix A.1.

Objective-wise advantage decoupling. Since directly combining multiple rewards before group normalization can obscure their individual learning signals, inspired by (Liu et al., 2026d), we instead normalize each objective independently within the rollout group:

$$A_i^j = \frac{r_i^j - \mu^j}{\sigma^j + \epsilon_{\text{norm}}}, \qquad j \in \{Q, C, L\}, \tag{9}$$

where μ^j and σ^j are the group-wise mean and standard deviation for objective j, and ϵ_{norm} is a small constant for numerical stability. We then aggregate the normalized advantages according to the preference $\boldsymbol{\lambda}$ and further normalize the combined signal at the batch level:

$$\bar{A}_i^{\boldsymbol{\lambda}} = \lambda_Q A_i^Q + \lambda_C A_i^C + \lambda_L A_i^L, \qquad A_i^{\boldsymbol{\lambda}} = \frac{\bar{A}_i^{\boldsymbol{\lambda}} - \mu_{\bar{A}}}{\sigma_{\bar{A}} + \epsilon_{\text{norm}}}, \tag{10}$$

where $\mu_{\bar{A}}$ and $\sigma_{\bar{A}}$ are the batch-wise mean and standard deviation. The resulting advantage is used in the following clipped GRPO loss:

$$\mathcal{L}_{\text{MemPilot}} = -\mathbb{E}_{i,k}\left[\min\left(\rho_{i,k} A_i^{\boldsymbol{\lambda}}, \text{clip}(\rho_{i,k}, 1-\epsilon_{\text{clip}}, 1+\epsilon_{\text{clip}}) A_i^{\boldsymbol{\lambda}}\right)\right] + \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}), \tag{11}$$

where $\rho_{i,k}$ is the probability ratio between the current policy π_θ and the rollout policy π_{old} for token k of trajectory i, ϵ_{clip} is the clipping coefficient, β controls the KL regularization strength, and π_{ref} denotes the reference policy. Varying $\boldsymbol{\lambda}$ yields policies with different quality–cost–latency trade-offs while preserving objective-specific learning signals.

Prefix-based marginal utility. Trajectory-level advantages provide the same learning signal across curation stages, obscuring their individual contributions. We therefore probe the rollout after each stage t by truncating subsequent curation and directly generating an answer $\hat{y}_{i,t}$. Let

$$u_{i,t} = Q(\hat{y}_{i,t}), \qquad \Delta u_{i,t} = u_{i,t} - u_{i,t-1}, \tag{12}$$

where $\Delta u_{i,t}$ estimates the marginal answer-quality improvement contributed by stage t. Since cost and latency are directly attributable and already captured at the trajectory level, prefix probing focuses only on answer quality to avoid double-counting resource penalties. We use this signal to refine credit assignment for the policy decisions associated with that stage:

$$\widetilde{A}_{i,k} = A_i^{\boldsymbol{\lambda}} + \eta\, \Delta u_{i,t}, \qquad k \in \mathcal{S}_{i,t}, \tag{13}$$

where $\mathcal{S}_{i,t}$ contains the policy reasoning and action tokens that precede the corresponding curation response, and η controls the strength of the marginal credit. Other tokens retain the trajectory-level advantage $A_i^{\boldsymbol{\lambda}}$. The stage-refined advantage $\widetilde{A}_{i,k}$ replaces $A_i^{\boldsymbol{\lambda}}$ in Eq. 11 for $k \in \mathcal{S}_{i,t}$, incurring little overhead due to short answer completions and largely shared prefix KV-cache reuse.

4 Experiment

4.1 Setups

Benchmarks and Metrics. We mainly evaluate MemPilot in multimodal settings involving both textual and visual interaction histories. Specifically, we use five multimodal agent-memory benchmarks: Mem-Gallery (Bei et al., 2026), WorldMemArena (Liu et al., 2026a), H2HMem (Zhu et al., 2026), MemEye (Guo et al., 2026), and MemLens (Ren et al., 2026), with MemEye and MemLens further serving as out-of-distribution evaluation sets. We evaluate answer quality using F1 score and an LLM-judge score (L-J), with GPT-4o-mini as the judge model. Cost measures the total token-based monetary cost incurred by both the orchestrator and delegated external models. Latency measures their total inference latency in seconds. Appendix A.1 and A.4 provide further details.

Baselines. We organize baselines into two groups according to whether they support explicit trade-off control. The first group contains fixed-configuration methods used for the main comparison. It includes *(1) text-centric memory systems* including **A-MEM** (Xu et al., 2025) and **SimpleMem** (Liu et al., 2026c), *(2) training-based memory methods* including **GAM** (Yan et al., 2025a), **AgeMem** (Yu et al., 2026b), and **Mem-T** (Yue et al., 2026), and *(3) multimodal memory systems* including **MIRIX** (Wang & Chen, 2025), **MemVerse** (Liu et al., 2025), and **M2A** (Feng et al., 2026a). The second group contains *(4) methods that explicitly expose performance–efficiency trade-offs*, including **LightMem** (Fang et al., 2025), **BudgetMem** (Zhang et al., 2026b), and **OmniSimpleMem** (Liu et al., 2026b). Section 4.4 compares these methods separately through frontier analysis.

4.2 Implementation Details

MemPilot is implemented based on VeRL[1], with Qwen3-4B-Instruct (Yang et al., 2025) as the default orchestrator policy. To disentangle gains from RL-enhanced answer generation, we additionally report a decoupled-answer evaluation where the curated context is passed to a separate, non-RL-trained answer model, using Qwen3-VL-4B-Instruct and Qwen3.5-9B. External LLMs and VLMs are accessed through OpenRouter[2], with the heterogeneous model pool specified in Appendix A.2. We segment raw interaction histories into 256-token chunks and use Qwen3-Embedding-0.6B (Zhang et al., 2025c) for semantic retrieval. Unless otherwise specified, the query-agnostic memory bank is constructed with LLMLingua-2 (Pan et al., 2024) at a 0.4 compression ratio for policy training and main evaluation. We further examine compatibility with memory banks produced by other agent memory systems in Section 4.6. We retrieve at most five evidence items per step for both MemPilot and applicable baselines. Since memory items from different systems can vary substantially in length, we directly compare monetary cost and latency rather than memory-item counts. Remaining hyperparameters are provided in Appendix A.5.

For training, we adopt a 7/1/2 train/validation/test split. We merge the training portions of Mem-Gallery, WorldMemArena, and H2HMem into a joint training set and evaluate in-domain performance on their held-out test sets. We further use MemLens and MemEye exclusively for out-of-domain evaluation to assess generalization to unseen memory distributions. The same data protocol is applied to all applicable baselines, with detailed dataset statistics provided in Appendix A.4. For the main comparison, we evaluate three representative MemPilot preferences: **Perf**, **Bal**, and **Cost**,

[1] https://github.com/volcengine/verl

[2] https://openrouter.ai

Table 1: **Overall comparison on Mem-Gallery, WorldMemArena, H2HMem, MemEye, and MemLens**. We report F1 score (**F1**), LLM-judge score (**L-J**), and inference cost (**Cost**), with latency analyzed separately in Section 4.4. Within each column, the best result is boldfaced and highlighted with a darker shade, while the second-best result is highlighted with a lighter shade.

Method	Mem-Gallery			WorldMemArena			H2HMem			MemEye†			MemLens†		
	F1↑	L-J↑	Cost↓	F1↑	L-J↑	Cost↓	F1↑	L-J↑	Cost↓	F1↑	L-J↑	Cost↓	F1↑	L-J↑	Cost↓
Answer Model: Qwen3-VL-4B-Instruct															
A-Mem	54.24	51.45	2.8e-2	21.38	47.73	1.8e-1	23.27	23.88	3.6e-2	15.20	20.28	1.3e-2	22.99	21.15	2.6e-2
SimpleMem	27.32	32.00	1.9e-2	15.58	26.36	4.5e-2	16.67	20.38	2.2e-2	14.69	16.87	**4.6e-3**	18.46	20.23	2.3e-3
MIRIX	36.70	49.45	3.3e-1	19.23	54.09	7.8e-1	19.81	25.56	3.8e-1	12.92	21.23	2.4e-1	15.91	13.73	8.4e-2
M2A	39.05	50.73	8.3e-1	20.14	54.49	1.1e+0	27.21	33.19	9.1e-1	12.26	17.86	8.5e-1	15.23	17.20	3.0e-1
MemVerse	30.92	44.27	4.7e-1	21.22	60.91	1.1e+0	20.03	26.04	6.0e-1	15.28	19.95	3.2e-1	18.99	16.91	1.4e-1
GAM	43.46	46.36	1.5e-2	13.33	35.68	3.9e-2	23.87	23.96	2.3e-2	12.04	19.64	2.4e-2	23.43	20.81	2.2e-3
AgeMem	38.38	39.36	3.0e-2	19.45	42.95	5.4e-2	16.46	16.11	3.6e-2	10.48	11.86	1.9e-2	15.22	14.19	2.5e-3
Mem-T	36.18	34.55	1.9e-2	21.14	43.27	7.0e-2	23.15	23.85	2.3e-2	**20.89**	22.78	9.0e-3	17.22	18.01	4.6e-3
MemPilot-Perf	53.73	**65.82**	4.3e-2	**22.01**	**63.41**	5.1e-2	**30.32**	**35.97**	6.0e-2	15.34	**25.07**	5.0e-2	**30.58**	**27.17**	5.5e-4
MemPilot-Bal	**57.26**	64.27	1.4e-2	20.71	50.00	1.3e-2	24.88	28.96	1.6e-2	13.46	22.37	1.0e-2	21.99	21.39	1.8e-4
MemPilot-Cost	32.67	39.09	**8.2e-3**	6.25	15.00	**7.0e-3**	15.18	14.58	**8.5e-3**	8.13	12.94	5.6e-3	7.63	6.36	**1.2e-4**
Answer Model: Qwen3.5-9B															
A-Mem	44.68	53.82	5.5e-2	20.33	54.09	2.6e-1	24.78	31.46	5.9e-2	13.04	19.21	2.2e-2	21.46	19.58	3.6e-2
SimpleMem	25.26	30.73	3.4e-2	10.65	19.77	6.0e-2	19.14	20.28	2.6e-2	11.27	14.69	**6.9e-3**	13.42	11.56	3.6e-3
MIRIX	37.13	50.64	3.3e-1	20.59	57.05	1.2e+0	22.80	31.94	6.9e-1	11.18	21.48	2.8e-1	20.34	22.11	1.1e-1
M2A	26.14	36.00	9.5e+0	16.26	42.50	7.6e+0	22.89	24.44	1.0e+1	10.41	13.95	6.1e+0	13.40	14.23	9.3e-1
MemVerse	21.08	42.45	5.0e-1	21.01	60.23	1.3e+0	20.91	32.85	5.2e-1	10.89	13.01	3.5e-1	17.89	17.63	1.8e-1
GAM	38.36	37.72	2.1e-2	20.51	37.50	5.4e-2	26.92	23.68	2.8e-2	14.13	19.56	1.5e-2	22.51	19.08	3.7e-3
AgeMem	33.48	34.55	4.4e-2	20.17	45.00	7.2e-2	17.41	18.33	4.9e-2	11.21	11.99	3.0e-2	16.35	14.45	4.1e-3
Mem-T	33.00	31.18	2.1e-2	21.72	44.09	7.9e-2	26.01	27.92	2.3e-2	12.88	19.05	1.3e-2	23.20	21.59	5.5e-3
MemPilot-Perf	48.19	**67.64**	5.0e-2	**23.55**	**74.09**	5.7e-2	**30.62**	**37.57**	6.7e-2	**15.52**	**21.83**	5.4e-2	**28.08**	**24.28**	6.5e-4
MemPilot-Bal	**53.55**	67.27	2.1e-2	20.98	59.32	1.9e-2	28.58	35.69	2.3e-2	15.26	18.87	1.5e-2	18.51	17.92	2.8e-4
MemPilot-Cost	29.74	38.55	**1.5e-2**	8.81	21.14	**1.3e-2**	14.43	16.39	**1.6e-2**	8.77	8.09	1.0e-2	8.08	6.36	**2.2e-4**

† denotes out-of-distribution (OOD) evaluation datasets.

prioritizing answer quality, balanced quality–resource trade-offs, and inference cost, respectively. Additional preference settings are explored in Section 4.4.

4.3 Main Results

As shown in Table 1, MemPilot spans distinct quality–cost regimes across both models and benchmarks. The results reveal several consistent trends across methods and settings. *(1) Text-centric systems achieve low cost largely by avoiding additional multimodal processing and runtime curation, but this often comes with lower answer quality.* In comparison, MemPilot-Bal generally achieves stronger overall quality at moderate cost, while MemPilot-Cost reaches similarly low-cost regimes on most datasets. *(2) Runtime adaptation does not necessarily translate into controllable trade-offs.* Although GAM, AgeMem, and Mem-T adapt memory processing at inference time, they largely expose a fixed operating regime rather than explicitly controlling the quality–cost balance. In contrast, MemPilot spans multiple operating points. MemPilot-Perf achieves the best L-J score across all five benchmarks under both models, while MemPilot-Bal substantially lowers cost and provides a distinct intermediate quality–cost trade-off. *(3) Multimodal memory processing can be costly without adaptive allocation.* MIRIX, M2A, and MemVerse often incur substantially higher inference cost without consistently translating that expense into better answer quality. In contrast, MemPilot selectively invokes external curation and achieves stronger performance at much lower cost in most settings, with the same trend extending to MemEye and MemLens. *Overall, MemPilot provides controllable quality–cost trade-offs rather than a single performance-oriented operating point.* MemPilot-Perf prioritizes answer quality, MemPilot-Bal offers a balanced operating point, and MemPilot-Cost further reduces resource usage with greater quality sacrifice. We analyze latency and finer-grained performance–cost–latency frontiers in Section 4.4.

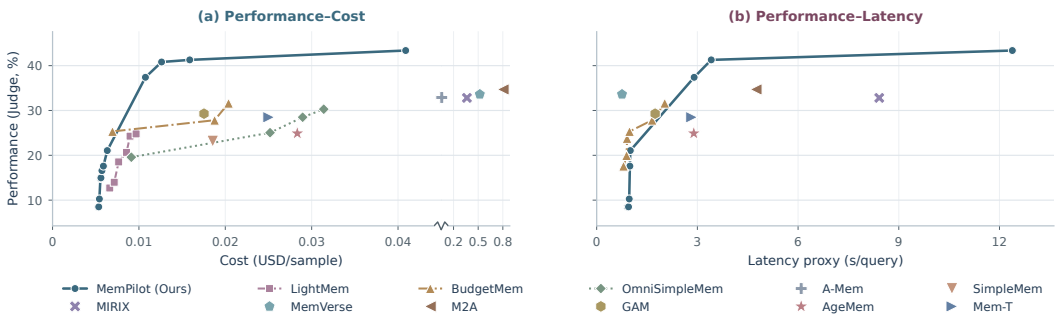


Figure 2: **Performance–cost–latency frontiers.** Results are macro-averaged over five benchmarks. Lines connect Pareto-optimal points, while faint markers show other evaluated settings. Operating points are obtained by varying optimization preferences or budget settings.

4.4 Exploring Performance–Cost–Latency Frontiers

We further examine fine-grained trade-off frontiers by sweeping MemPilot's optimization preferences. In addition to the main baselines, we include LightMem (Fang et al., 2025), BudgetMem (Zhang et al., 2026b), and OmniSimpleMem (Liu et al., 2026b), which explicitly support performance–efficiency trade-offs. For latency, we report representative fixed-inference baselines together with runtime-adaptive methods, since most query-agnostic systems exhibit nearly unchanged inference behavior and collapse to similar single operating points.

Performance–cost. As shown in Figure 2(a), MemPilot forms a stronger frontier across a broad cost range and largely envelopes the competing methods. Compared with trade-off-aware baselines such as LightMem, BudgetMem, and OmniSimpleMem, MemPilot reaches higher quality as additional computation is allocated, while also covering competitive low-cost operating points. Most conventional memory systems expose only a fixed quality–cost point, highlighting the additional flexibility provided by learned runtime orchestration. Multimodal systems such as MIRIX, M2A, and MemVerse generally incur much higher cost due to visual processing, whereas MemPilot reduces unnecessary multimodal computation through on-demand curation and selective model delegation.

Performance–latency. Figure 2(b) shows a similarly broad quality–latency frontier. MemPilot spans lightweight to performance-oriented configurations, whereas most baselines expose only a narrow region or a single operating point. Even among runtime-adaptive methods, the achievable latency range remains limited. MemVerse achieves competitive quality at relatively low latency, but lacks explicit trade-off control and incurs substantially higher monetary cost in Figure 2(a). This further highlights that cost and latency capture complementary resource constraints.

Overall, MemPilot provides a broader, more controllable performance–cost–latency frontier through adaptive runtime orchestration, enabling flexible computation allocation under resource preferences.

4.5 How Does MemPilot Allocate Runtime Computation?

To better understand how MemPilot realizes different trade-offs, we analyze its learned runtime curation behavior in Figure 3. Panels (a) and (b) show that as the optimization preference shifts toward lower cost or lower latency, the policy adjusts multiple control dimensions in a coordinated manner rather than relying on a single knob. Moving from left to right, MemPilot generally reduces the number of policy steps, the frequency of **CURATE** actions, image access, and policy output tokens. At the same time, evidence count also changes adaptively, indicating that the policy does not simply retrieve less information, but learns to reconfigure how memory is accessed and processed under different resource preferences.

Panel (c) further shows that model selection is a key mechanism behind this adaptation. Under higher-cost settings, MemPilot relies more heavily on stronger VLMs and LLMs, while lower-cost settings shift computation toward cheaper model groups. Together, these results show that MemPilot learns a coherent allocation strategy over curation depth, modality access, and routed model capacity, which explains its ability to realize controllable performance–cost–latency trade-offs.

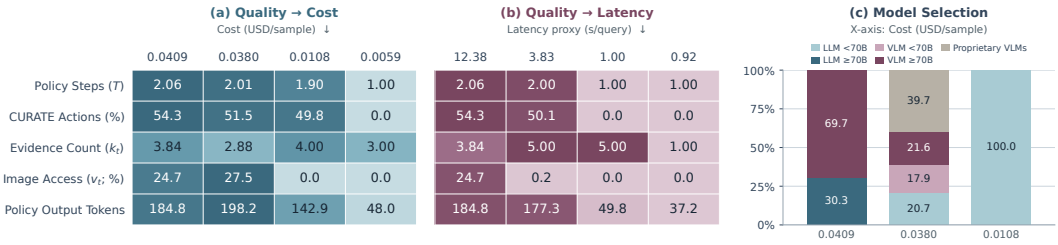


Figure 3: **Allocation of runtime computation under different optimization preferences.** Panels (a,b) start with the same quality-oriented policy, with cost and latency proxy decreasing from left to right, respectively. Panel (c) shows model-selection shares across model groups. Values are averaged across five benchmarks; darker cells indicate larger values within each row.

4.6 Discussion

Ablation Study of Core Design Choices. We ablate four key components of MemPilot to examine their contributions to runtime orchestration and policy optimization, with Qwen3-VL-4B-Instruct as an answer model. As shown in Table 2, removing LLM/VLM delegation reduces the average Judge score from 37.40 to 14.58, highlighting the importance of adaptive external curation. Removing marginal utility or retrieval–instruction decoupling causes similarly large degradation, demonstrating the value of fine-grained credit assignment and separating evidence retrieval from curation instructions. Objective-wise advantage decoupling is also important for multi-objective training, with its removal reducing the Judge score to 21.94. Although all ablated variants consume less inference cost, their substantial quality losses indicate that simply reducing runtime computation does not yield a desirable trade-off.

Table 2: **Ablation study.** Results are averaged across five benchmarks.

Variant	L-J ↑	Cost ↓
MemPilot-Bal	**37.40**	1.1e-2
w/o LLM/VLM Delegation	14.58	5.6e-3
w/o Marginal Utility	14.82	5.5e-3
w/o GDPO	21.94	5.9e-3
w/o Retrieval–Instruction Decoupling	15.35	5.4e-3

Compatibility with Off-the-Shelf Memory Banks. To examine whether MemPilot depends on a particular query-agnostic memory construction, we replace its default memory bank with off-the-shelf agent memory banks and evaluate the performance–cost frontiers without retraining using Qwen3-VL-4B-Instruct as an answer model. Figure 4 reports averages across five benchmarks. The resulting frontiers are clearly separated: the default bank occupies the lower-cost regime, SimpleMem provides an intermediate profile, while A-MEM incurs higher cost but reaches stronger performance. Each memory bank thus induces a distinct operating regime, while MemPilot improves performance as additional runtime computation is allocated. This demonstrates compatibility with diverse upstream memory systems while preserving their distinct performance–cost characteristics through runtime orchestration.

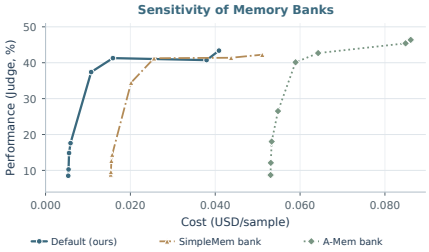


Figure 4: **Memory-bank compatibility.** Performance–cost frontiers with different memory-bank initializations.

5 Conclusion

We presented **MemPilot**, a framework for orchestrating on-demand memory curation under controllable performance–cost–latency preferences. MemPilot formulates runtime memory adaptation as a multi-step decision process, combines query-agnostic memory with raw interaction history, and adaptively delegates curation to heterogeneous LLMs and VLMs. We further introduced resource-aware policy optimization with objective-wise advantage decoupling and prefix-based marginal utility for fine-grained credit assignment. Experiments across multimodal agent-memory benchmarks demonstrate that MemPilot achieves favorable trade-off frontiers while adapting computation to different optimization preferences. We hope this work encourages agent memory systems to move beyond performance-only optimization toward resource-aware runtime adaptation.

AI USE STATEMENT

In this work, we used generative AI tools to assist with language polishing and the search for relevant literature. All AI-assisted text and retrieved references were manually reviewed and verified by the authors. We take full responsibility for the final content of this work, including its text, claims, results, and citations.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the MemPilot framework, policy optimization, and reward formulation in the main methodology section. Experimental settings, evaluation protocols, dataset statistics, model-pool configurations, reward computation, and hyperparameters are documented in Appendix A.4, Appendix A.2, Appendix A.1, and Appendix A.5. We will also release our source code upon publication to facilitate reproduction and future research.

REFERENCES

Yuanchen Bei, Tianxin Wei, Xuying Ning, Yanjun Zhao, Zhining Liu, Xiao Lin, Yada Zhu, Hendrik Hamann, Jingrui He, and Hanghang Tong. Mem-gallery: Benchmarking multimodal long-term conversational memory for mllm agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 40750–40784, 2026.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. Routerdc: Query-based router by dual contrastive learning for assembling large language models. *Advances in Neural Information Processing Systems*, 37:66305–66328, 2024.

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.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks Lakshmanan, and Ahmed H Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. In *International Conference on Learning Representations*, volume 2024, pp. 41348–41366, 2024.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, et al. Lightmem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*, 2025.

Junyu Feng, Binxiao Xu, Jiayi Chen, Mengyu Dai, Cenyang Wu, Haodong Li, Bohan Zeng, Yunliu Xie, Hao Liang, Ming Lu, et al. M2a: Multimodal memory agent with dual-layer hybrid memory for long-term personalized interactions. *arXiv preprint arXiv:2602.07624*, 2026a.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. *arXiv preprint arXiv:2410.03834*, 2024.

Tao Feng, Fangxu Yu, Haozhen Zhang, Zhongjie Dai, Liangqi Yuan, Zijie Lei, Weizhi Zhang, Kunlun Zhu, Haodong Yue, Keyang Xuan, et al. Llmrouter: Unified infrastructure for developing, evaluating, and deploying llm routers. *arXiv preprint arXiv:2608.06867*, 2026b.

Tao Feng, Haozhen Zhang, Zijie Lei, Peixuan Han, and Jiaxuan You. Graphplanner: Graph memory-augmented agentic routing for multi-agent llms. *arXiv preprint arXiv:2604.23626*, 2026c.

Minghao Guo, Qingyue Jiao, Zeru Shi, Yihao Quan, Boxuan Zhang, Danrui Li, Liwei Che, Wujiang Xu, Shilong Liu, Zirui Liu, et al. Memeye: A visual-centric evaluation framework for multimodal agent memory. *arXiv preprint arXiv:2605.15128*, 2026.

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.

Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, et al. Rethinking memory mechanisms of foundation agents in the second half: A survey. *arXiv preprint arXiv:2602.06052*, 2026.

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

Chengzhi Liu, Yuzhe Yang, Sophia Xiao Pu, Yepeng Liu, Lin Long, Yichen Guo, Nuo Chen, Zhaotian Weng, Elena Kochkina, Simerjot Kaur, et al. Worldmemarena: Evaluating multimodal agent memory through action-world interaction. *arXiv preprint arXiv:2605.29341*, 2026a.

Jiaqi Liu, Zipeng Ling, Shi Qiu, Yanqing Liu, Siwei Han, Peng Xia, Haoqin Tu, Zeyu Zheng, Cihang Xie, Charles Fleming, et al. Omni-simplemem: Autoresearch-guided discovery of lifelong multimodal agent memory. *arXiv preprint arXiv:2604.01007*, 2026b.

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

Junming Liu, Yifei Sun, Weihua Cheng, Haodong Lei, Yirong Chen, Licheng Wen, Xuemeng Yang, Daocheng Fu, Pinlong Cai, Nianchen Deng, et al. Memverse: Multimodal memory for lifelong learning agents. *arXiv preprint arXiv:2512.03627*, 2025.

Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Peter Belcak, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, et al. Gdpo: Group reward-decoupled normalization policy optimization for multi-reward rl optimization. *arXiv preprint arXiv:2601.05242*, 2026d.

Lin Long, Yichen He, Wentao Ye, Yiyuan Pan, Yuan Lin, Hang Li, Junbo Zhao, and Wei Li. Seeing, listening, remembering, and reasoning: A multimodal agent with long-term memory. In *International Conference on Learning Representations*, volume 2026, pp. 146197–146246, 2026.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: Towards llms as operating systems. 2023.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981, 2024.

Cheng Qian, Zuxin Liu, Shirley Kokane, Akshara Prabhakar, Jielin Qiu, Haolin Chen, Zhiwei Liu, Heng Ji, Weiran Yao, Shelby Heinecke, et al. xrouter: Training cost-aware llms orchestration system via reinforcement learning. *arXiv preprint arXiv:2510.08439*, 2025.

Xiyu Ren, Zhaowei Wang, Yiming Du, Zhongwei Xie, Chi Liu, Xinlin Yang, Haoyue Feng, Wenjun Pan, Tianshi Zheng, Baixuan Xu, et al. Memlens: Benchmarking multimodal long-term memory in large vision-language models. *arXiv preprint arXiv:2605.14906*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Yu Wang and Xi Chen. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*, 2025.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-{\alpha}: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 2025.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

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

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Kristian Kersting, Jeff Z Pan, Hinrich Schütze, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025b.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Jianing Yin and Tan Tang. Defermem: Query-time evidence distillation via reinforcement learning for long-term memory qa. *arXiv preprint arXiv:2605.22411*, 2026.

Jing Yu, Yibo Zhao, Jiaming Zhang, and Xiang Li. Lazymem: Retrieve broadly, construct selectively for efficient long-term agent memory. *arXiv preprint arXiv:2607.22690*, 2026a.

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

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572, 2025.

Yanwei Yue, Boci Peng, Xuanbo Fan, Jiaxin Guo, Qiankun Li, and Yan Zhang. Mem-t: Densifying rewards for long-horizon memory agents. *arXiv preprint arXiv:2601.23014*, 2026.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents. *arXiv preprint arXiv:2509.24704*, 2025a.

Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026a.

Haozhen Zhang, Haodong Yue, Tao Feng, Quanyu Long, Jianzhu Bao, Bowen Jin, Weizhi Zhang, Xiao Li, Jiaxuan You, Chengwei Qin, et al. Learning query-aware budget-tier routing for runtime agent memory. *arXiv preprint arXiv:2602.06025*, 2026b.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026c.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025c.

Yunxiang Zhang, Yiheng Li, Ali Payani, and Lu Wang. Adamem: Test-time adaptive memory for language agents. *arXiv preprint arXiv:2606.05684*, 2026d.

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*, volume 38, pp. 19724–19731, 2024.

Shiping Zhu, Yibo Yang, Zhengyang Wang, Tiancheng Shen, Dandan Guo, and Ming-Hsuan Yang. H2hmem: A multimodal memory benchmark for agents in human-human interactions. *arXiv preprint arXiv:2606.09461*, 2026.

APPENDIX CONTENTS

A More Implementation Details **14**

A.1 Detailed Reward Modeling 14

A.2 Heterogeneous Model Pool 16

A.3 Prefix-Based Marginal-Utility Credit Assignment 16

A.4 Evaluation Details 17

A.5 Hyperparameters 19

B More Experimental Results **19**

B.1 Per-Benchmark Runtime Computation Allocation 19

B.2 Per-Benchmark Ablation Results 21

B.3 Memory-Bank Compatibility Across Benchmarks 21

C Case Study **21**

D Limitations and Future Work **23**

E Broader Impact **23**

F Prompts **24**

F.1 Policy Prompt 24

F.2 Delegation Prompt 25

F.3 Question-Answering Prompt 25

F.4 Answer-Judging Prompts 27

APPENDIX

A MORE IMPLEMENTATION DETAILS

A.1 DETAILED REWARD MODELING

Cost Reward. We account for the inference cost of both the locally hosted orchestrator and the external models. Let $\mathcal{S}_\pi(\tau)$ denote the set of orchestrator-generation segments in trajectory τ, and let $\mathcal{E}(\tau)$ denote its external-model calls. The monetary cost in USD is

$$
\begin{aligned}
C_\pi(\tau) &= 10^{-6} \sum_{s \in \mathcal{S}_\pi(\tau)} \left(p_\pi^{\text{in}} n_{\pi,s}^{\text{in}} + p_\pi^{\text{out}} n_{\pi,s}^{\text{out}} \right), \\
C_{\text{ext}}(\tau) &= 10^{-6} \sum_{c \in \mathcal{E}(\tau)} \left(p_{m_c}^{\text{in}} n_c^{\text{in}} + p_{m_c}^{\text{out}} n_c^{\text{out}} \right), \\
C(\tau) &= C_\pi(\tau) + C_{\text{ext}}(\tau),
\end{aligned}
\tag{14}
$$

where m_c is the external model selected for call c, and all prices are specified in USD per million tokens. We assign the orchestrator proxy prices $p_\pi^{\text{in}} = 0.05$ and $p_\pi^{\text{out}} = 0.25$ for `Qwen3-4B-Instruct`. The external-model prices p_m^{in} and p_m^{out} are listed in Table 3. When the API returns an exact monetary cost, we use it in place of the static token-based estimate.

For orchestrator accounting, the initial prompt is charged once. Each policy-generated token is counted once as output, while newly appended environment tokens, including observation responses,

are counted as incremental inputs to the next step generation. Previously processed prompt and assistant-history tokens are not charged again, corresponding to ideal within-trajectory KV-prefix reuse. A direct **RETRIEVE** call incurs no external-model cost, although the orchestrator generations before and after that call remain included.

Latency Reward. We use a deterministic latency proxy rather than the observed wall-clock latency of individual API requests, which can vary substantially with network conditions and transient serving load. LLM inference latency is roughly influenced by fixed per-request overhead, input-dependent prefill computation, and autoregressive output decoding. Rather than modeling the full hardware- and serving-dependent latency profile, we adopt a deliberately simplified affine proxy that summarizes these effects with a fixed overhead and average per-token contributions for input and output over the operating range considered in our experiments. Accordingly, for an orchestrator-generation segment s, we estimate its latency as

$$\widehat{L}_{\pi,s} = b_\pi + a_\pi^{\text{in}} n_{\pi,s}^{\text{in}} + a_\pi^{\text{out}} n_{\pi,s}^{\text{out}}, \tag{15}$$

where b_π captures the fixed invocation and scheduling overhead, while a_π^{in} and a_π^{out} model the marginal latency of input prefill and autoregressive decoding, respectively. The orchestrator coefficients are

$$\left(b_\pi, a_\pi^{\text{in}}, a_\pi^{\text{out}}\right) = \left(0.332,\, 3.0{\times}10^{-6},\, 6.64{\times}10^{-3}\right), \tag{16}$$

measured in seconds, seconds per input token, and seconds per output token, respectively. We apply the same incremental-token accounting used for monetary cost: the initial prompt is counted once, each newly generated orchestrator segment contributes output tokens, and only newly inserted environment content contributes additional input tokens.

For an external-model call c routed to model m_c, we estimate

$$\widehat{L}_c = b_{m_c} + a_{m_c}^{\text{in}} n_c^{\text{in}} + a_{m_c}^{\text{out}} n_c^{\text{out}} + a_{m_c}^{\text{img}} n_c^{\text{img}}, \tag{17}$$

where the image term is zero for text-only LLMs. Table 3 reports the model-specific coefficients.

Calls emitted in the same assistant turn are executed in parallel and form one latency stage. We therefore take the maximum latency within each parallel stage and sum across sequential stages:

$$\widehat{L}_{\text{ext}}(\tau) = \sum_{t=1}^{T} \max_{c \in \mathcal{E}_t(\tau)} \widehat{L}_c, \tag{18}$$

where $\mathcal{E}_t(\tau)$ contains the external calls made in stage t. The total trajectory latency is

$$\widehat{L}(\tau) = \sum_{s \in \mathcal{S}_\pi(\tau)} \widehat{L}_{\pi,s} + \widehat{L}_{\text{ext}}(\tau). \tag{19}$$

We calibrate the external-model profiles using repeated non-streaming OpenRouter requests. Each model is evaluated through a fixed backend route with provider fallback disabled, and cache-busting request content is used to avoid identical-request cache hits. Controlled short-input, long-input, long-output, and image-input cases isolate the fixed, per-input-token, per-output-token, and per-image terms. We use median measurements to reduce sensitivity to transient outliers and constrain all coefficients to be non-negative.

Overall. Let $Q(\tau) \in [0,1]$ denote the task-specific answer-quality score and let $g(\tau) \in \{0,1\}$ indicate whether the trajectory satisfies the complete output-format contract. We apply the format gate to obtain the task reward:

$$r^{\text{task}}(\tau) = \begin{cases} Q(\tau), & g(\tau) = 1, \\ -1, & g(\tau) = 0. \end{cases} \tag{20}$$

Thus, a format-valid trajectory receives its answer-quality score, whereas a format-invalid trajectory receives a fixed reward of -1.

The format gate checks that the trajectory contains exactly one non-empty final `<answer>...</answer>` block with no trailing output, that every action has balanced tags and parseable JSON arguments, and that all required action fields are present. Action-side

validation failures, such as invalid model or memory identifiers and overlength arguments, also invalidate the trajectory. Since memory evidence is required for the evaluated tasks, producing a final answer without invoking either RETRIEVE or CURATE is also treated as invalid.

Format-invalid and padding trajectories are excluded from both cost and latency comparison. Their resource rewards and resource advantages are set to zero, preventing a malformed or no-tool trajectory from obtaining a favorable resource signal merely because it incurs little cost or latency. For a format-valid trajectory, however, resource rewards remain active even when $Q(\tau) = 0$; cost and latency are therefore not multiplied or gated by answer performance. This preserves the independence of the three optimization objectives.

The resulting task, cost, and latency components are normalized and combined at the advantage level as described in Section 3.3. Prefix-based marginal-utility probes use the ungated answer-quality score because their synthetic outputs intentionally omit the main trajectory's action structure. Probe generations are used only for credit assignment and are excluded from monetary cost, latency, and external-call statistics.

A.2 HETEROGENEOUS MODEL POOL

We list the heterogeneous model pool in Table 3. All models are accessed through OpenRouter[3]. For each model m, p_m^{in} and p_m^{out} denote the reference input and output prices in USD per million tokens. The latency profile is represented as $\boldsymbol{\phi}_m = (b_m, a_m^{\text{in}}, a_m^{\text{out}})$ for text-only LLMs and $\boldsymbol{\phi}_m = (b_m, a_m^{\text{in}}, a_m^{\text{out}}, a_m^{\text{img}})$ for VLMs. Here, b_m is the base latency in seconds per call, while a_m^{in}, a_m^{out}, and a_m^{img} are measured in seconds per input token, output token, and input image, respectively. These coefficients are fixed latency proxies rather than unstable live wall-clock measurements. Their use and calibration are detailed in Section A.1.

Table 3: **Heterogeneous model pool used in our experiments**. Input and output prices, denoted by p_m^{in} and p_m^{out}, are in USD per million tokens. The latency profile is ordered as $(b_m, a_m^{\text{in}}, a_m^{\text{out}})$ for LLMs and $(b_m, a_m^{\text{in}}, a_m^{\text{out}}, a_m^{\text{img}})$ for VLMs.

Model	p_m^{in}	p_m^{out}	Latency profile $\boldsymbol{\phi}_m$
Text-only LLMs			
`Llama-3.2-3B-Instruct`	0.05	0.33	$(1.231,\ 4.8\times10^{-5},\ 5.54\times10^{-3})$
`Llama-3.1-8B-Instruct`	0.05	0.08	$(0.439,\ 3.6\times10^{-5},\ 4.99\times10^{-3})$
`Llama-3.3-70B-Instruct`	0.13	0.40	$(0.475,\ 9.2\times10^{-5},\ 2.194\times10^{-2})$
`Qwen3-30B-A3B-Instruct-2507`	0.10	0.30	$(0.524,\ 4.5\times10^{-5},\ 5.33\times10^{-3})$
`Qwen3-Next-80B-A3B-Instruct`	0.10	1.10	$(0.517,\ 5.2\times10^{-5},\ 1.126\times10^{-2})$
`Qwen3-235B-A22B-2507`	0.09	0.55	$(0.797,\ 5.0\times10^{-5},\ 5.88\times10^{-2})$
Vision-language models			
`Qwen3-VL-8B-Instruct`	0.117	0.455	$(0.332,\ 3.0\times10^{-6},\ 6.64\times10^{-3},\ 0.225)$
`Qwen3-VL-30B-A3B-Instruct`	0.15	0.60	$(0.438,\ 6.6\times10^{-5},\ 5.327\times10^{-2},\ 0.052)$
`Qwen3-VL-235B-A22B-Instruct`	0.21	1.90	$(0.588,\ 8.5\times10^{-5},\ 5.293\times10^{-2},\ 1.722)$
`Gemini-2.5-Flash-Lite`	0.10	0.40	$(0.920,\ 6.7\times10^{-5},\ 3.97\times10^{-3},\ 1.097)$
`Claude-Haiku-4.5`	1.00	5.00	$(1.189,\ 3.5\times10^{-5},\ 1.151\times10^{-2},\ 0.904)$
`GPT-5.6-Luna`	0.50	3.00	$(1.697,\ 8.8\times10^{-5},\ 8.09\times10^{-3},\ 0.282)$

A.3 PREFIX-BASED MARGINAL-UTILITY CREDIT ASSIGNMENT

Algorithm 1 provides the implementation details of the prefix-based credit assignment introduced in Section 3.3. For a trajectory τ_i containing K_i causal curation stages, we record the pre-curation prefix $C_{i,0}$ and the prefix $C_{i,t}$ after the evidence returned by stage t has been incorporated. Multiple curation operations initiated concurrently at the same decision point are treated as one stage because their returned evidence is observed jointly.

[3] `https://openrouter.ai/`

Using the same rollout policy, we greedily generate a short answer from each prefix without performing any subsequent memory curation. These $K_i + 1$ probes yield answer-quality scores $u_{i,0}, \ldots, u_{i,K_i}$, from which the signed marginal utility of each stage is computed as

$$\Delta u_{i,t} = u_{i,t} - u_{i,t-1}. \tag{21}$$

The resulting credit is added only to the policy reasoning and action tokens associated with that stage. The probes are auxiliary training computations and are excluded from monetary-cost, latency, and delegated-model invocation accounting. They are submitted to the same rollout engine, allowing automatic prefix caching to reuse shared prefixes when available.

Algorithm 1 Prefix-based marginal-utility credit assignment.

Require: Completed rollout τ_i with K_i causal curation stages
Require: Prefixes $\{C_{i,t}\}_{t=0}^{K_i}$ and policy-token spans $\{\mathcal{S}_{i,t}\}_{t=1}^{K_i}$
Require: Ground-truth answer y_i, trajectory-level advantage $A_i^{\boldsymbol{\lambda}}$, marginal weight η, and probe-length limit $T_{\max}$
Ensure: Stage-refined token advantages $\{\widetilde{A}_{i,k}\}$

$\widetilde{A}_{i,k} \leftarrow A_i^{\boldsymbol{\lambda}}$ for every policy token k in τ_i
if $K_i = 0$ **or** τ_i is format-invalid **then**
  return $\{\widetilde{A}_{i,k}\}$
end if
for $t = 0, \ldots, K_i$ **in parallel do**
  $\hat{y}_{i,t} \leftarrow \textsc{GreedyAnswer}(\pi_{\text{old}}, C_{i,t} \,\|\, \texttt{<answer>}, T_{\max})$ ▷ No subsequent memory curation
  $u_{i,t} \leftarrow Q(\hat{y}_{i,t}, y_i)$
end for
if any answer probe fails **then**
  return $\{\widetilde{A}_{i,k}\}$
end if
for $t = 1, \ldots, K_i$ **do**
  $\Delta u_{i,t} \leftarrow u_{i,t} - u_{i,t-1}$
  for all $k \in \mathcal{S}_{i,t}$ **do**
    $\widetilde{A}_{i,k} \leftarrow \widetilde{A}_{i,k} + \eta\, \Delta u_{i,t}$
  end for
end for
return $\{\widetilde{A}_{i,k}\}$

Here, $C_{i,0}$ provides the pre-curation baseline required to estimate the contribution of the first curation stage.

A.4 Evaluation Details

Memory preprocessing and indexing. We serialize each multimodal conversation in chronological order and partition it into approximately 256-token text chunks. Dialogue turns are treated as atomic units: a turn is never split across chunks, and its attached images, image identifiers, and captions remain associated with the same chunk. An oversized individual turn is retained as a single chunk. For WorldMemArena (Liu et al., 2026a), the memory available to each question is additionally restricted to the cumulative conversation prefix visible at its corresponding checkpoint.

We construct separate dense indexes for the raw and compressed memory corpora. Query-agnostic compression is performed independently for chunk text and image captions using LLMLingua-2 (Pan et al., 2024) with a default compression rate of 0.4, while memory and image identifiers remain unchanged. Document embeddings are computed offline using Qwen3-Embedding-0.6B (Zhang et al., 2025c), normalized, stored in float16 format, and reused throughout evaluation. At runtime, only the policy-generated retrieval query is encoded, after which evidence is ranked by embedding similarity. The compressed-memory and raw-memory indexes are searched independently, with at most five retrieved items per curation operation. Image captions participate in textual retrieval, whereas image pixels are made available only when an image-capable external model is selected. Reference answers are never used during preprocessing, indexing, or retrieval.

Benchmarks and data splits. Our in-distribution evaluation covers Mem-Gallery (Bei et al., 2026), the lifelong subset of WorldMemArena (Liu et al., 2026a), and both the dyadic and multiparty variants of H2HMem (Zhu et al., 2026). Each dataset is split at the conversation level into 70% training, 10% validation, and 20% test data, ensuring that questions derived from the same conversation never appear in different splits. WorldMemArena is additionally stratified by category and scenario. The source datasets are split independently before their training and validation portions are merged, and their source identities are preserved for dataset-wise reporting.

For out-of-distribution evaluation, we use the complete open-answer portion of MemEye (Guo et al., 2026), containing 371 questions across eight scenarios, and the official 32K memory-agent subset of MEMLENS (Ren et al., 2026). MemEye's multiple-choice variant is excluded. MEMLENS answer-refusal questions are removed, leaving 173 of the 195 agent-subset questions. Answer-refusal questions are likewise excluded from Mem-Gallery and H2HMem. For H2HMem, we additionally remove dialogues without usable memory content and Conflict Detection instances whose reference labels cannot satisfy the benchmark's required standalone Yes/No answer format. Neither OOD benchmark contributes data to policy training or model selection.

Table 4: **Benchmark statistics after applying our data-integrity and answer-refusal filters**. An instance denotes an independently partitioned memory history; for MEMLENS, each question-specific 32K history constitutes one instance. IID datasets are split at the instance level, whereas OOD benchmarks are used exclusively for testing.

		Train		Validation		Test	
Benchmark	Setting	#Inst.	#QA	#Inst.	#QA	#Inst.	#QA
Mem-Gallery	IID	14	1,062	2	190	4	275
WorldMemArena-Lifelong	IID	27	1,485	3	165	8	440
H2HMem-Dyadic	IID	13	1,122	2	174	4	327
H2HMem-Multiparty	IID	3	97	1	33	1	33
IID total		**57**	**3,766**	**8**	**562**	**17**	**1,075**
MemEye-Open	OOD	–	–	–	–	8	371
MEMLENS-32K-Agent	OOD	–	–	–	–	173	173
OOD total		–	–	–	–	**181**	**544**

Inference. We evaluate one deterministic trajectory per question using greedy decoding with temperature 0. The maximum prompt and response lengths are both 4,096 tokens. Each trajectory permits at most six sequential curation rounds and at most two concurrent independent curation operations. All policy-facing memory-access settings are kept identical to those used during the corresponding training run.

LLM-as-a-judge evaluation. We additionally evaluate semantic answer correctness using `openai/gpt-4o-mini`. We preserve each benchmark's official judge role, correctness rubric, score semantics, and output convention. The only sample-dependent inputs presented to the judge are the predicted answer and the ground-truth reference answer.

Minor answer-only adaptations are applied when an official prompt requires information unavailable to common baseline outputs. Specifically, we remove the question field from all judge inputs; remove Key Memory Points from WorldMemArena because they evaluate evidence coverage rather than final-answer correctness; convert H2HMem's session-batched evaluation into itemwise evaluation; and remove the previous answer from MEMLENS Knowledge Update prompts because correctness is determined by the current reference answer. Mem-Gallery and H2HMem use five-level scores in $\{0, 0.25, 0.5, 0.75, 1\}$. WorldMemArena classifies each answer as CORRECT, HALLUCINATION, or OMISSION, with only CORRECT receiving score 1. MemEye follows its official score quantization to $\{0, 0.5, 1\}$, and MEMLENS uses type-specific binary rubrics. Judge-model token usage is retained for audit but excluded from the system's cost, latency, and invocation metrics.

Policy–answer model decoupling. For controlled comparisons that separate memory orchestration from answer-generation ability, we optionally remove the trained policy's final answer while retaining the query and the memory evidence collected along its curation trajectory. This information is

provided to a separately specified, untrained answer model using a benchmark-specific QA prompt. Policy reasoning, routing arguments, system instructions, and reference answers are excluded from this prompt. The replacement model's cost and latency are added to the corresponding resource totals, but its common final-answer invocation is not counted as a routed external-model invocation.

Aggregation and resource reporting. F1, EM, LLM-judge score, and QA latency are macro-averaged over questions. Let $\mathcal{Q}$ denote the evaluated questions and let N_{sample} be the number of unique dataset–conversation pairs. We aggregate resource metrics as

$$\text{Cost}_{\text{QA}} = \frac{\sum_{q\in\mathcal{Q}} C(\tau_q)}{N_{\text{sample}}}, \qquad \text{Latency}_{\text{QA}} = \frac{\sum_{q\in\mathcal{Q}} \widehat{L}(\tau_q)}{|\mathcal{Q}|}, \qquad \#\text{Calls}_{\text{QA}} = \frac{\sum_{q\in\mathcal{Q}} N_{\text{ext}}(\tau_q)}{N_{\text{sample}}}, \tag{22}$$

where $N_{\text{ext}}(\tau_q)$ counts only attempted routed external-LLM invocations. It excludes local policy generation, retrieval-only operations, judge-model requests, optional common answer-model generation, and offline memory construction. QA cost includes policy inference, routed external models, and the optional decoupled answer model. Section A.1 provides detailed cost and latency definitions.

A.5 Hyperparameters

We summarize the hyperparameters used in our main experiments in Table 5.

Table 5: **Hyperparameters used in the main experiments**.

Hyperparameter	Value
Policy model	`Qwen3-4B-Instruct-2507`
Embedding model	`Qwen3-Embedding-0.6B`
Maximum retrieved items per call	5
Rollouts per query G	4
Rollout temperature	1.0
Training batch size	32
Learning rate	1×10^{-6}
Training epochs	3
Maximum prompt length	4,096 tokens
Maximum response length	4,096 tokens
Maximum context length	8,192 tokens
Maximum user/assistant turns	6 / 6
Maximum parallel tool calls	2
Maximum external-model output	512 tokens
External-model temperature	0.0
GRPO clipping coefficient ϵ_{clip}	0.2
KL-loss coefficient β	0.001
Marginal-utility weight η	0.5

B More Experimental Results

B.1 Per-Benchmark Runtime Computation Allocation

Figure 5 provides the per-benchmark breakdown of the runtime-allocation analysis in Section 4.5. The overall behavior is consistent across all five benchmarks. As stronger cost or latency preferences are imposed, MemPilot generally reduces policy steps, **CURATE** actions, visual access, and policy-generation tokens, indicating coordinated adaptation across multiple control dimensions.

Model routing exhibits an equally consistent shift. High-resource settings rely heavily on larger LLMs and VLMs, whereas tighter cost preferences progressively move curation toward lighter models. At the lowest displayed cost point, all five benchmarks route exclusively to LLMs below 70B. Meanwhile, individual controls such as evidence depth and image access need not change monotonically, showing that MemPilot reallocates computation jointly rather than uniformly shrinking every

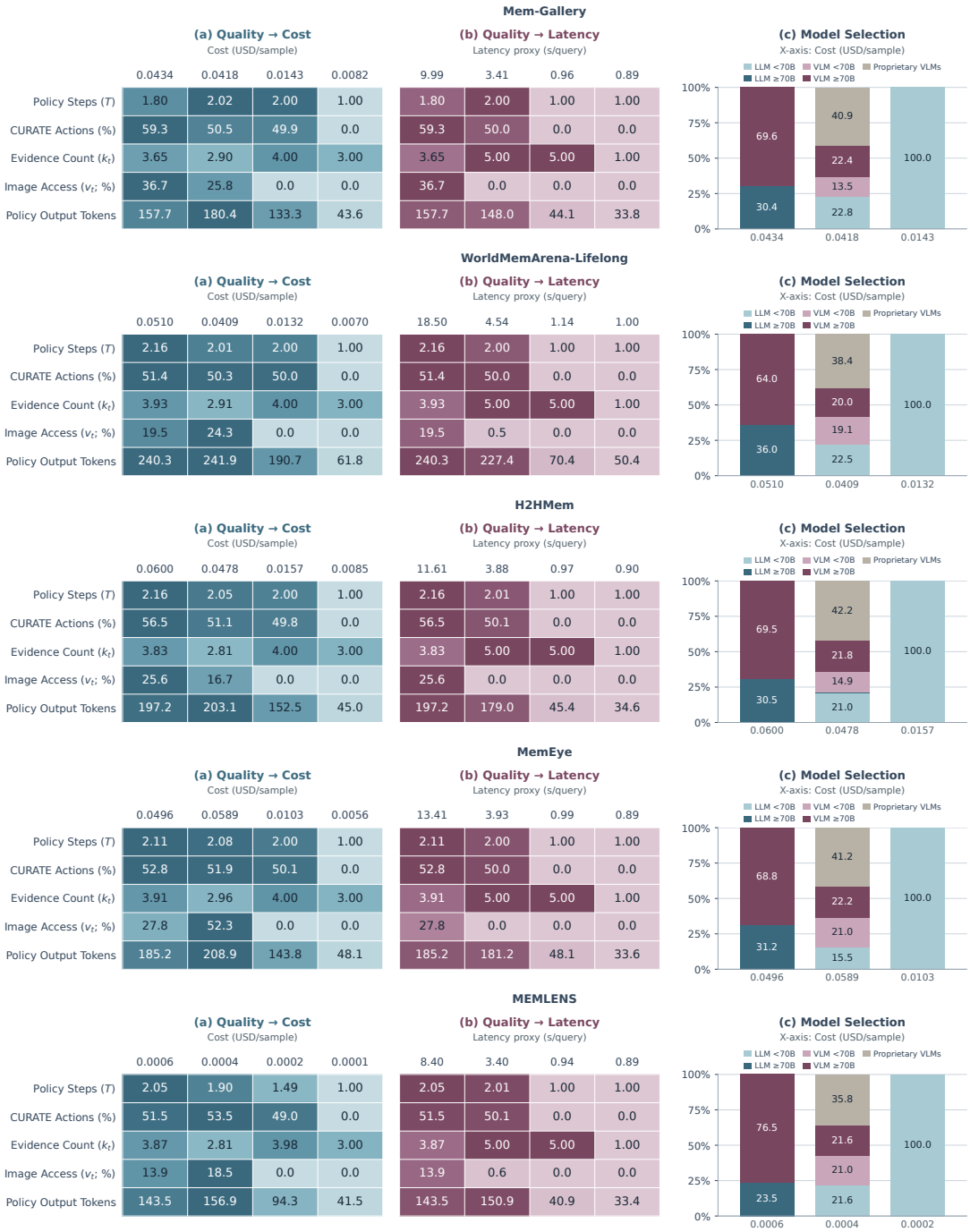


Figure 5: **Per-benchmark runtime computation allocation.** Runtime curation behaviors under varying cost and latency preferences on Mem-Gallery, WorldMemArena, H2HMem, MemEye, and MemLens, respectively from top to bottom.

decision dimension. These results confirm that the aggregate behavior observed in the main text is broadly consistent across datasets while retaining dataset-specific curation patterns.

Table 6: **Per-benchmark ablation results.** Absolute Judge scores and inference costs are reported for each variant.

Variant	Mem-Gallery		WorldMemArena		H2HMem		MemLens†		MemEye†	
	L-J ↑	Cost ↓	L-J ↑	Cost ↓	L-J ↑	Cost ↓	L-J ↑	Cost ↓	L-J ↑	Cost ↓
MemPilot-Bal	**64.27**	1.4e-2	**50.00**	1.3e-2	**28.96**	1.6e-2	**21.39**	1.8e-4	**22.37**	1.0e-2
w/o LLM/VLM Delegation	32.00	8.1e-3	9.09	6.6e-3	12.08	8.2e-3	6.94	1.1e-4	12.80	5.2e-3
w/o Marginal Utility	31.18	7.6e-3	9.77	6.5e-3	17.57	7.9e-3	3.47	1.1e-4	12.13	5.1e-3
w/o GDPO	41.91	7.8e-3	21.36	7.3e-3	20.69	8.3e-3	9.83	1.1e-4	15.90	5.6e-3
w/o Retrieval–Instruction Decoupling	34.45	7.5e-3	10.91	6.4e-3	16.04	7.8e-3	3.47	1.1e-4	11.86	5.0e-3

† denotes out-of-distribution (OOD) evaluation.

B.2 Per-Benchmark Ablation Results

Table 6 provides the per-benchmark breakdown of the ablation study. The full MemPilot configuration consistently achieves the highest Judge score across all five benchmarks. Among the ablations, removing objective-wise advantage decoupling generally preserves the strongest performance, yet still causes clear degradation on every dataset, confirming its role in multi-objective policy optimization. Removing LLM/VLM delegation, marginal utility, or retrieval–instruction decoupling leads to substantially larger drops, with similar trends observed on both in-domain and out-of-domain benchmarks. Although all ablations reduce inference cost, the accompanying quality degradation remains pronounced, indicating that the complete design yields a more effective quality–cost trade-off rather than merely consuming additional computation.

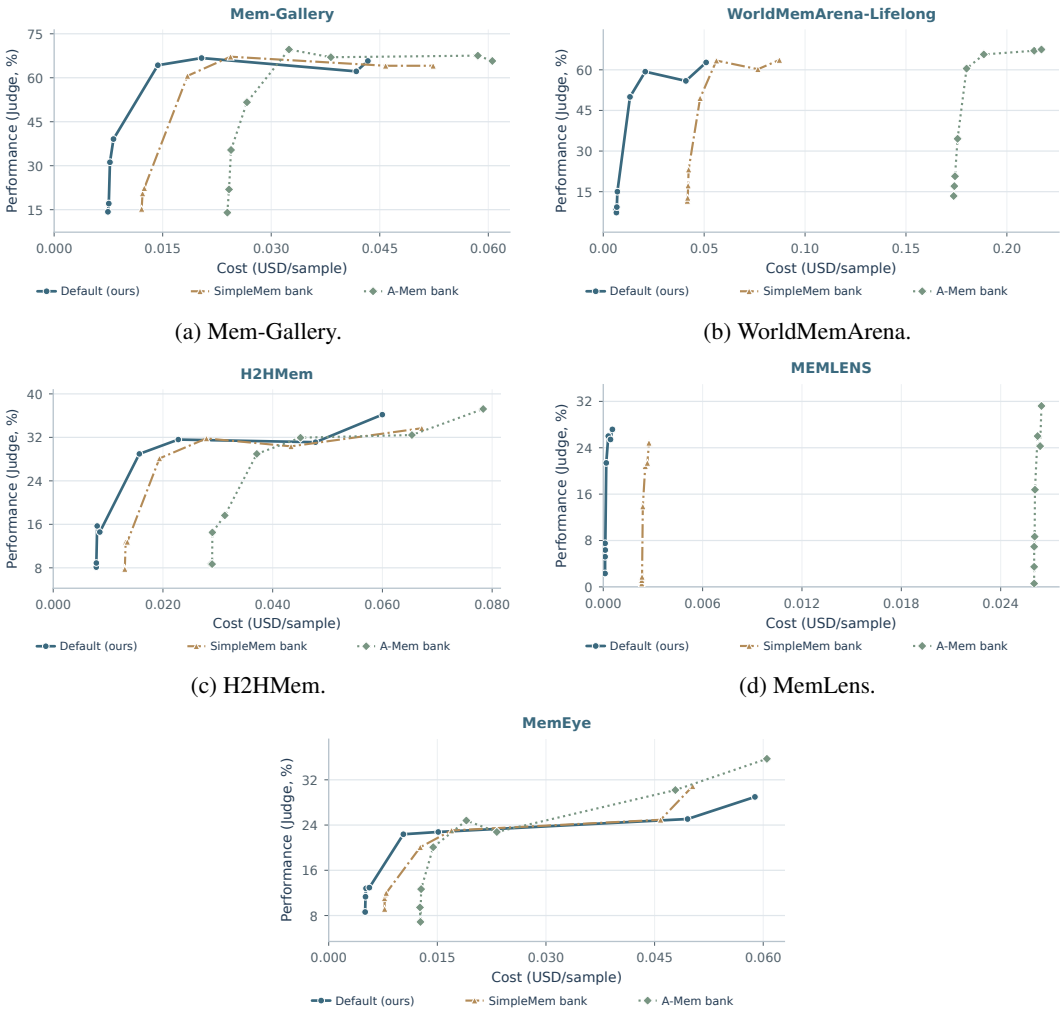
B.3 Memory-Bank Compatibility Across Benchmarks

We further report per-benchmark results for the memory-bank compatibility analysis in Figure 6. Across all five benchmarks, replacing the default memory bank with SimpleMem (Liu et al., 2026c) or A-MEM (Xu et al., 2025) preserves clear performance–cost frontiers without retraining, confirming that MemPilot is not tightly coupled to a particular query-agnostic memory construction. Different memory banks occupy distinct cost regimes while retaining similar improvements as runtime computation increases. The default bank generally provides more cost-efficient operating points, whereas A-MEM incurs higher overall cost but often reaches a higher performance ceiling, particularly on WorldMemArena, MemEye, and MemLens. SimpleMem typically lies between these regimes and remains competitive on Mem-Gallery and H2HMem. These results suggest that MemPilot preserves its runtime orchestration behavior across diverse memory banks while retaining the distinct performance–cost characteristics of their upstream memory representations.

C Case Study

Case 1: Direct Retrieval without Delegation

Question. Which image shows a picnic mat? Only return the image_id of the image.
Step 1: Retrieve compressed memory. `retrieve_memory`
`retrieval_query`: image of a picnic mat
`evidence_count`: 1
Tool response (excerpt). `image_id`: `D13:IMG_001`
`image_caption`: red white cloth background resembling picnic blanket tablecloth square pattern curved edges
[Other retrieved text omitted.]
Final answer. `<answer>D13:IMG_001</answer>`
Reference answer. `D13:IMG_001`



(a) Mem-Gallery.

(b) WorldMemArena.

(c) H2HMem.

(d) MemLens.

(e) MemEye.

Figure 6: **Per-benchmark memory-bank compatibility.** Performance–cost frontiers of MemPilot with different memory-bank initializations without retraining across five benchmarks.

Case 2: Text-Only Delegation for Counting

Question. How many Button-Down Chambray Dress options did Lin Wei show? Please answer using an Arabic numeral (e.g., "2" instead of "Two").
Step 1: Retrieve compressed memory. `retrieve_memory`
`retrieval_query`: Button-Down Chambray Dress shown by Lin Wei
`evidence_count`: 3
Tool response (abridged). The returned compressed memory references dress images `D8:IMG_001`, `D8:IMG_002`, and `D8:IMG_005`, alongside discussion of the dress options.
[Full compressed-memory response omitted.]
Step 2: Delegate raw-memory processing. `runtime_memory`
`retrieval_query`: Button-Down Chambray Dress shown by Lin Wei
`instruction`: Count the number of Button-Down Chambray Dress options shown by Lin Wei in the retrieved memory.
`model`: `model_7ab3` (3B text-only LLM)
`evidence_count`: 3
`include_images`: `false`

Tool response (abridged). The external model enumerates five dress options, corresponding to `D8:IMG_001`, `D8:IMG_002`, `D8:IMG_003`, `D8:IMG_004`, and `D8:IMG_005`.
[Descriptions of individual dresses omitted.]
Final answer. `<answer>5</answer>`
Reference answer. 5

Case 3: Expanded Retrieval with Image-Enabled Delegation

Question. Which Butterfly Sleeve Sheath Dress did Lin Wei decide to buy. Only return the image_id of the image.
Step 1: Retrieve compressed memory. `retrieve_memory`
`retrieval_query`: Butterfly Sleeve Sheath Dress purchased by Lin Wei
`evidence_count`: 1
Tool response (excerpt). D8 : 16
: winners : chambray # 3 butterfly sleeve # 2 shopping detour
[Other retrieved text omitted; this response contains no image ID for the selected dress.]
Step 2: Expand retrieval and enable image input. `runtime_memory`
`retrieval_query`: Butterfly Sleeve Sheath Dress purchased by Lin Wei
`instruction`: Identify the image_id of the Butterfly Sleeve Sheath Dress that Lin Wei decided to buy based on the provided memory content.
`model`: `model_9a74` (VLM; 235B total, 22B active)
`evidence_count`: 5
`include_images`: `true`
Tool response (abridged). The external model associates the first dress with `D8:IMG_006` and the second, dark dress with red abstract patterns with `D8:IMG_007`. It connects Lin Wei's choice of "butterfly sleeve #2" to `D8:IMG_007`.
[Remaining external-model response omitted.]
Final answer. `<answer>D8:IMG_007</answer>`
Reference answer. `D8:IMG_007`

D Limitations and Future Work

While MemPilot provides flexible runtime memory orchestration, several aspects could be further improved:

- **Privacy-aware delegation.** Runtime curation may send retrieved interaction history to external models, which can raise privacy concerns for sensitive memories. Future work could incorporate privacy-aware routing, local processing, or access-control mechanisms while remaining orthogonal to MemPilot's core orchestration design.
- **Preference calibration.** MemPilot controls trade-offs through predefined optimization preferences. Future work could study more user-friendly ways to map high-level deployment preferences to reward coefficients, making trade-off specification easier across applications.
- **Deployment-aware latency modeling.** We use a proxy latency estimator during training to avoid instability from transient network and serving fluctuations. Future work could instead optimize directly against measured latency in a controlled system-level sandbox with stable networking and serving conditions.

E Broader Impact

MemPilot advocates a broader shift in agent memory from performance-oriented optimization toward explicit and controllable performance–cost–latency trade-offs. In practical deployments, the highest-performing memory configuration is not always the most desirable, as different users and applications may place substantially different values on monetary cost and response latency. This mirrors modern commercial LLM services, which increasingly offer model tiers spanning different capability, cost, and latency profiles (e.g., GPT-5.6-Sol/Terra/Luna), allowing users to choose

operating points that best match their needs. We believe agent memory systems should provide similar flexibility rather than committing to a single performance-oriented configuration. This becomes particularly relevant for multimodal agents, where accessing and processing visual evidence can introduce substantially different resource demands. More broadly, treating memory curation as adaptive computation allocation may encourage tighter integration between agent memory, model routing, and resource-aware decision making, enabling future long-horizon agents to become more efficient, responsive, and deployment-aware.

F PROMPTS

F.1 POLICY PROMPT

The policy shares the following instructions across benchmarks. Benchmark-specific inputs and answer requirements are represented by placeholders. Tool schemas specify valid arguments and model IDs. The tools `retrieve_memory` and `runtime_memory` implement Retrieve and CURATE, respectively.

Policy Prompt Template

Role. You are a runtime-memory agent that solves questions by planning and executing an adaptive memory pipeline, directly retrieving compressed evidence or delegating raw memory chunks to capability-diverse external LLMs for processing as needed.
Input (user message). `<question>`, with `<benchmark-specific metadata and answer constraints>` when applicable.
Instruction. Work iteratively:

- Use `retrieve_memory` for faster access to precomputed query-agnostic compressed memory, which may omit important details. Use `runtime_memory` to delegate query-aware processing to a selected external LLM using the complete, uncompressed content of each retrieved raw memory chunk, while incurring some additional model-call cost and latency.
- For visual content, `retrieve_memory` exposes only textual captions and cannot inspect image pixels. To inspect pixels from retrieved or question images, use `runtime_memory` with an image-capable LLM and set `include_images` to `true`.
- Before the first call, decompose the query into one or more atomic, evidence-dependent subtasks. Execute only the next unresolved subtask; do not delegate multiple dependent subtasks through one broad instruction.
- Before each tool call, briefly assess what is known, what remains unresolved, and why the next call is useful. After each memory-tool result, update this assessment before deciding on another call or the final answer.
- Execute dependent subtasks sequentially. Calls in the same assistant message run in parallel, so put only independent subtasks in separate `tool_call` blocks, at most two per message. If a later call depends on an earlier memory-tool result, issue it in a later turn.

Evidence constraint. Each successful tool result contains findings supported by that call's inputs and may identify unresolved parts. Missing evidence is local to the selected inputs, not proof that the full query is unanswerable. Use partial findings to plan later calls or the final answer.
Tool-call format. Available call forms (serialization only; neither is preferred):
Direct compressed-memory retrieval:

```
<tool_call>
{"name": "retrieve_memory",
  "arguments": {
    "retrieval_query":
      "<self-contained evidence query>",
    "evidence_count": <integer>
  }}
</tool_call>
```

Routed original-memory processing:

```
<tool_call>
{"name": "runtime_memory",
 "arguments": {
   "retrieval_query":
     "<self-contained evidence query>",
   "instruction":
     "<evidence-processing instruction>",
   "model": "<model ID>",
   "evidence_count": <integer>,
   "include_images": true
 }}
</tool_call>
```

Choose every argument value for the current step.
Output. Once you have enough evidence, stop calling tools and finish with `<benchmark-specific final-answer contract>`. Do not write anything after the closing `</answer>` tag.

F.2 Delegation Prompt

The selected LLM/VLM receives the policy-generated instruction and retrieved raw evidence. The retrieval query is used for search and is not separately included in this prompt. Image-related sections are optional; image pixels are attached when requested and supported by the selected model.

Delegation Prompt Template

System. You are a memory-processing assistant. Follow the instruction using only the question-image descriptions, memory items, and images in the input. Return all findings supported by these inputs. If the instruction can only be partially completed, return the supported findings and briefly state what remains unresolved. If nothing relevant is present, say so directly. Do not add unsupported information.

User.
Instruction:
`<policy-generated evidence-processing instruction>`
Question image descriptions:
`<question-image IDs and captions, if present>`
Memory items:
`<retrieved raw memory chunks with IDs and metadata>`
Additional image inputs:
`<selected memory-image IDs and captions, if present>`

F.3 Question-Answering Prompt

The answer model receives the question and memory evidence returned by the policy, without the policy's reasoning or original final answer. The shared template uses the benchmark-specific instructions listed below.

QA Prompt Template

System. `<benchmark-specific QA instructions>`
Return only one non-empty `<answer>...</answer>` block, with the answer text inside the tags.

User.
`<memory heading>:`
`[Evidence 1]`
`<memory evidence returned by a policy tool>`
`[Evidence 2]`
`<additional memory evidence>`

...
Question Image:
<question-image IDs and captions, if available>
Question Date:
<question date, if provided for MemLens>
Question:
<question>
<question-specific output constraint, if applicable>

The following instructions replace the benchmark-specific placeholder in the system message.

QA Instructions: Mem-Gallery

Answer from the provided multimodal conversation memory. Ground the answer in that memory, prefer the latest information when details change over time, and keep the response concise but complete. When the question asks for images, preserve the exact image_id values.

Additional user-message constraint for conflict detection:
The text inside <answer>...</answer> must be exactly "Yes." or "No."

Additional user-message constraint for visual search:
Return the exact matching image_id value or values. If several apply, sort them in ascending order and separate them with commas.

QA Instructions: WorldMemArena

Answer from the provided retrieved memories. Synthesize multiple entries when needed, use timestamps and the most recent evidence to resolve conflicts, and ground every claim in the memories. Convert relative dates using the memory timestamps. When images are requested, preserve exact image_id values. Keep the answer concise; if no relevant information is present, the answer text must be exactly "Not mentioned in memory."

QA Instructions: H2HMem

You are a memory testing system. <question-type instruction> Answer directly in English without reasoning, using no more than 100 words.

Additional system instruction for conflict detection:
The answer text must be strictly either Yes or No.

Additional system instruction for answer refusal:
If the information is absent, the answer text must be exactly "Not mentioned."

For H2HMem, the question-type instruction is selected from the following:

H2HMem Question-Type Instructions

Unimodal precise recall. Accurately recall the requested information from the conversation memory.
Cross-modal related retrieval. Use related textual and visual information from the conversation memory.
Knowledge resolution. Resolve knowledge consistently across the conversation memory.
Temporal reasoning. Reason about temporal relationships in the conversation memory.
Multimodal causal reasoning. Use textual and visual evidence for the requested causal reasoning.
Reference & evolution tracking. Track references and how they evolve across the conversation memory.
Test-time learning. Use knowledge introduced in the conversation memory to answer the question.

Conflict detection. Determine whether the information in the question conflicts with the conversation memory.
Answer refusal. Determine whether the question can be answered from the conversation memory.

QA Instructions: MemEye (Open-Ended)

Answer from the provided long-horizon multimodal conversation memory. Ground the answer in the memory and relevant visual evidence; when evidence conflicts, use the newest visual evidence unless the question itself asks about the conflict. For counts, give the number; for yes/no questions, begin with Yes or No; keep descriptive answers to one or two sentences. If the requested information is absent, state that it is unavailable.

QA Instructions: MemLens

Answer from the provided conversation history. If the question cannot be answered from it, the answer text must be exactly "Insufficient information".

F.4 Answer-Judging Prompts

Mem-Gallery and H2HMem. These benchmarks share the following five-level rubric. In the output-format example, the reasoning placeholder is empty for Mem-Gallery and `<short explanation>` for H2HMem.

Judge Prompt: Mem-Gallery and H2HMem

You are an impartial judge evaluating the memory capabilities of an AI assistant with the question-answering task.
Your task is to compare the Assistant's Answer against the Ground Truth and assign a score of 0, 0.25, 0.5, 0.75, or 1.
Scoring Rubric.
Score 0 (Incorrect / Miss):

- The answer contradicts the Ground Truth.
- For Yes/No questions: The answer has the wrong polarity (e.g., says "Yes" when Ground Truth is "No").
- For Open-ended questions: The answer provides factually wrong information or hallucinations.
- The assistant fails to provide the required information.

Score 0.25 (Poor / Tangential):

- The answer touches on the topic but misses the **core entity** or key value required.
- The answer contains a mix of minor correct details and **significant hallucinations** or wrong associations.
- The answer is excessively vague to the point of being useless (e.g., answering "a dog" instead of "a golden retriever").

Score 0.5 (Partial / Vague):

- The answer is technically correct, but lacks confidence or is incomplete.
- The answer captures the **main entity or concept** correctly but misses a part of the required supporting details.
- For Yes/No questions: The polarity is correct, but the reasoning is flawed (if have), or the assistant is uncertain (e.g., "I think it might be Yes").
- For Open-ended questions: The answer is too general or misses key adjectives/details present in the Ground Truth.

Score 0.75 (Good / Minor Imperfection):

- The answer is largely accurate and captures the core information confidently.
- It misses only **minor details** (e.g., specific adjectives or secondary details) that do not alter the main truth.
- The answer contains all the correct information but includes unnecessary "fluff" or slight conversational filler that reduces precision.

Score 1 (Correct / Exact):

- The answer is accurate, precise, and confident.
- For Yes/No questions: The polarity matches the Ground Truth perfectly.
- For Open-ended questions: The answer contains **all** the core information and necessary details required by the Ground Truth without hallucinations.

Input Data.
Ground Truth: `<reference answer(s)>`
Assistant Answer: `<predicted answer>`
Output Format.
Output strictly in the following JSON format:

```
{"score": <0, 0.25, 0.5, 0.75, or 1>,
 "reasoning": "<reasoning placeholder>"}
```

WorldMemArena. The evaluator assigns a score of 1 to `Correct` and 0 to `Hallucination` or `Omission`. This evaluates answer correctness, not evidence coverage.

Judge Prompt: WorldMemArena

You are an **evaluation expert for AI memory system question answering**.
Based **only** on the provided **Reference Answer**, strictly evaluate the **accuracy** of the **Memory System Response**. Classify it as one of **Correct**, **Hallucination**, or **Omission**. Do **not** use any external knowledge or subjective inference.
Evaluation Criteria.
1. Correct

- The response accurately answers the question and is **semantically equivalent** to the Reference Answer.
- No contradictions with the Reference Answer.
- Synonyms, paraphrasing, and reasonable summarization are acceptable.

2. Hallucination

- The response includes information that **contradicts** the Reference Answer.
- When the Reference Answer is *unknown/uncertain*, yet the response provides a specific fact.

3. Omission

- The response is **incomplete** compared to the Reference Answer.
- It states "don't know" or "no related memory" even though the Reference Answer supplies the answer.
- For multi-element answers, missing **any** element counts as Omission.

Priority Rules.

- Both missing info AND fabricated info → **Hallucination**.
- No fabrication but missing info → **Omission**.
- Fully equivalent → **Correct**.

Information.
Reference Answer: `<reference answer(s)>`
Memory System Response: `<predicted answer>`
Output.

```
{
  "reasoning":  "Concise evaluation rationale",
  "evaluation_result":
    "Correct | Hallucination | Omission"
}
```

MemEye. The judge produces a five-level raw score. The evaluator subsequently maps raw scores $0, 0.25, 0.5, 0.75, 1$ to $0, 0.5, 0.5, 1, 1$, respectively. This mapping is post-processing and is not part of the prompt.

Judge Prompt: MemEye (Open-Ended)

You are an impartial judge evaluating the memory capabilities of an AI assistant on a question-answering task.
Your task is to compare the Assistant's Answer against the Ground Truth and assign a score of 0, 0.25, 0.5, 0.75, or 1.
Important Principles.

1. **Semantic equivalence over surface form.** The assistant may use different wording, synonyms, or sentence structure. As long as the meaning matches the Ground Truth, it should receive full credit. Do NOT penalize for stylistic differences.
2. **Numeric / counting questions are binary.** If the Ground Truth is a specific number (e.g., "3"), the answer is either exactly right (score 1) or wrong (score 0). There is no partial credit for being off by one.
3. **Negation + correction questions.** Some Ground Truth answers first negate a premise and then provide the correct information (e.g., "No — it was May 18-22, not May 15-19"). The assistant must get BOTH the negation AND the corrected fact right for full credit.
4. **Identity / "Who is" questions.** The Ground Truth may describe a person by role, actions, or relationships rather than by name. The assistant's answer is correct if it identifies the same unique person, even if it uses a name instead of a description or vice versa.
5. **Answer the thing that was asked.** Judge whether the assistant answered the requested attribute or type of information. If the question asks for a category, role, relationship, time, count, ordering, or specific item, an answer giving a different type of information is not fully correct even if it is related.
6. **Wrong entity beats related entity.** If the answer names the wrong person, object, brand, place, event, or option, do not give high scores just because it is on-topic. Being from the right semantic domain is not enough.
7. **Multi-item questions require set matching.** Full credit requires all required items and no incorrect extras. If only some required items are correct, partial credit may be appropriate.
8. **Ordering / chronology questions require the correct item.** If the question asks which item was first, last, middle, earlier, later, before, or after, the selected item must match the Ground Truth. A different plausible item is still wrong.
9. **Do not over-reward fluent explanations.** Extra detail is helpful only if it stays consistent with the Ground Truth. Confident wording or elaboration should not increase the score when the core fact is wrong.
10. **Contradicted relations or orderings are usually zero.** If the question asks about a relationship such as before/after, earlier/later, first/last, middle, sequence, relative timing, or other ordering, and the assistant gives the opposite relation, denies the relation, or selects the wrong ordered item, this is usually a clear contradiction and should normally receive score 0 rather than 0.25.

Scoring Rubric.
Score 0 (Incorrect / Miss):

- The answer contradicts the Ground Truth.
- For Yes/No questions: The answer has the wrong polarity (e.g., says "Yes" when Ground Truth is "No").
- For numeric questions: The number is wrong.
- For attribute-type mismatches: the answer gives the wrong kind of thing (for example, a brand when the question asks for a product category, or a related event when the question asks for the earliest specific item).
- For relation/order questions: if the answer contradicts the required before/after, earlier/later, first/last, middle, or similar relationship, score 0 in the usual case.
- The answer provides factually wrong information, hallucinations, or fails to provide any relevant information.

Score 0.25 (Poor / Tangential):

- The answer touches on the topic but misses the **core entity** or key value required.
- The answer contains a mix of minor correct details and **significant hallucinations** or wrong associations.
- The answer is excessively vague to the point of being useless (e.g., answering "a dog" instead of "a golden retriever").
- For multi-item questions: use 0.25 when the answer is clearly about the right topic but most of the required set is wrong.

Score 0.5 (Partial / Vague):

- The answer captures the **main entity or concept** correctly but misses important supporting details.
- For negation + correction questions: The polarity is correct but the corrected fact is missing or wrong, or vice versa.
- The answer is too general or misses key qualifiers present in the Ground Truth.
- For multi-item questions: use 0.5 when the answer gets a substantial part of the required set right (for example, one of two required items is correct and the rest is wrong or missing).

Score 0.75 (Good / Minor Imperfection):

- The answer is largely accurate and captures the core information confidently.
- It misses only **minor details** (e.g., specific adjectives or secondary details) that do not alter the main truth.
- The answer contains all the correct information but includes unnecessary filler that reduces precision.

Score 1 (Correct / Exact):

- The answer is accurate, precise, and confident.
- For Yes/No questions: The polarity matches the Ground Truth perfectly.
- For numeric questions: The number matches exactly.
- For descriptive questions: The answer contains **all** the core information and necessary details required by the Ground Truth without hallucinations.
- Semantic equivalence is sufficient — exact wording is NOT required.
- For multi-item questions: all required items are present and there are no incorrect extra items.

Input Data.
Ground Truth: `<reference answer(s)>`
Assistant Answer: `<predicted answer>`
Output Format.
Output strictly in the following JSON format:

```
{"score": <0, 0.25, 0.5, 0.75, or 1>,
  "reasoning": "<short explanation>"}
```

MemLens. MemLens uses a binary rubric with a task-specific criterion. Exactly one of the criteria listed below is inserted into the shared template.

Judge Prompt: MemLens

Now your role is a grading teacher. Your task is to review and score student answers based on reference standard answers for a question-answering benchmark.
First, extract the final answer from the student's solution, then judge whether the answer is correct. Score only the final answer; intermediate problem-solving steps do not need to be correct. If multiple inconsistent answers are given, or no clear final answer is stated, assign 0 points. Only the student's last clearly stated answer counts.
An item is covered if it is strictly mentioned or unambiguously implied by semantic equivalence. This includes numerical equivalence, synonyms, plural/singular forms, and equivalent date formats. Do not accept loosely related concepts. Ignore minor formatting differences, capitalization, punctuation, and equivalent wording when meaning is unchanged.
The scoring scale has two levels: 1 point and 0 points. Assign 1 point if the student's final answer matches the standard answer under the task-specific criteria below. Assign 0 points otherwise, including refusal or a claim of insufficient information except for answer-refusal tasks.
[Task-Specific Criteria]:
`<criterion selected for the question type>`
Your output format is:

```
[Scoring Rationale]:
[Score]: x points
[JSON]: {"answer_score": <0 or 1>}
```

[Current Case]
Standard Answer: `<reference answer(s)>`
Student Answer: `<predicted answer>`
[Scoring Rationale]:

MemLens Task-Specific Criteria

[IE — Entity / Attribute Extraction]
This is an information extraction question about a concrete entity, count, attribute, text, object, location, or visual detail from the provided memory context.

- Assign 1 point if the student's response contains the core information from the standard answer. Minor wording differences are acceptable, but the essential information must be present and correct.
- Assign 0 points if the core information is missing, contradicted, too vague, refused, or incorrect.
- For numeric or count answers, require the exact value unless the standard answer itself is approximate.

[IE — Previous Information Extraction]
This is an information extraction question about previously mentioned information, spatial relations, counts, or attributes from earlier conversation/image context.

- Assign 1 point if the student's response recovers the same previous information as the standard answer, including the correct relation, location, count, or attribute.
- Minor wording differences are acceptable if they preserve the same meaning.
- Assign 0 points if the answer refers to the wrong previous item, gives the wrong relation/count/location, is too vague, refuses, or is unsupported.

[KU — Knowledge Update]
This is a knowledge update question testing whether the model tracks the latest state.

- Assign 1 point if the student gives the current (correct) answer.
- Assign 0 points if the student gives an outdated answer, any other incorrect answer, refuses, or does not clearly answer.

[MSR — Yes/No (Entity Resolution)]
This is a yes/no question.

- Assign 1 point if the student's final answer is semantically equivalent to the standard answer (e.g., "Yes" matches "Yes", "No" matches "No"), even if phrased differently (e.g., "That's correct" for "Yes").
- Assign 0 points if the answer means the opposite, refuses, or does not clearly answer.

[MSR — Counting]
This is a counting question. The standard answer is a specific number.

- Assign 1 point if the student clearly states the exact number in any equivalent form (e.g., "3", "three", "3.0").
- Assign 0 points if the student gives a different number, a range, an approximation, refuses, or does not clearly answer.

[MSR — Arithmetic]
This is an arithmetic question. The standard answer is a specific value.

- Assign 1 point if the student gives the same value in an equivalent format (e.g., "0.5" and "1/2" and "50%").
- Assign 0 points if the student gives a different value, an unsupported approximation, refuses, or does not clearly answer.

[TR — Duration Comparison (A/B)]
This is a binary choice question. The student must select one of two options.

- Assign 1 point if the student clearly selects the same option as the standard answer, regardless of phrasing.
- Assign 0 points if the student selects the other option, refuses, or does not make a clear selection.

[TR — Order Ranking]
This is an ordering question. The standard answer is an exact sequence of items.

- Assign 1 point ONLY if the student gives the exact same sequence in the same order.
- Assign 0 points if any item is missing, added, duplicated, misplaced, or if the order is unclear.

[TR — Date Extraction]
This is a date extraction question. The standard answer is a specific date.

- Assign 1 point if the student gives the same date in any unambiguous equivalent format (e.g., "Jan 15, 2024" and "2024-01-15" and "15/01/2024").
- Assign 0 points if the date is different, incomplete when a full date is required, ambiguous, or missing.

[AR — Answer Refusal]

- Assign 1 point if the student explicitly refuses, says there is not enough information, or indicates the question is unanswerable from the provided context.
- Assign 0 points if the student gives a substantive answer, even if hedged.