MemPilot: Learning Transferable Latent Memory Mechanisms for LLM Reasoning

Changlong Shi[1], **Linhao Luo**[2], **Shigeng Chen**[3], **Guibin Zhang**[4], **Yi Chang**[5], **Shirui Pan**[3]*, **Chengqi Zhang**[1]
[1]The Hong Kong Polytechnic University, Hong Kong SAR, China
[2]Monash University, Australia [3]Griffith University, Australia
[4]National University of Singapore, Singapore [5]Jilin University, China
changlong.shi@connect.polyu.hk, s.pan@griffith.edu.au

Abstract

Integrating external memory into Large Language Models (LLMs) typically faces a trade-off between flexibility and depth. Explicit text-based retrieval keeps memory editable, but often acts as a static prompt prefix, increasing context length and interacting with the reasoning process only indirectly. Conversely, parametric memory can influence internal computation more deeply, but updating memory content usually requires additional optimization and may couple domain-specific information with the model's general reasoning behavior. In this work, we introduce MemPilot, a decoupled latent memory framework that dynamically guides the hidden reasoning states of frozen LLMs. MemPilot constructs an external bank of compact latent representations from offline reasoning trajectories, retrieves candidate memory entries, and integrates their latent memory tokens into hidden states via cross-attention and gated residual fusion. By separating domain-specific memory content from the learned memory-use mechanism, MemPilot enables target-domain adaptation through offline memory-bank substitution without updating the base model. Experiments across QA, coding, and mathematical reasoning benchmarks show that MemPilot achieves strong in-domain performance and more robust cross-domain transfer than other memory-augmented baselines. These results suggest that LLMs can benefit from reusable latent memory-use mechanisms while keeping memory content external and replaceable.

1 Introduction

In recent years, LLM-based agents have attracted substantial attention[1–3] and have been applied to domains such as personal assistance[4] and academic research[5]. As agents increasingly handle long-horizon decision-making, tool use, and interaction with external environments, they need memory mechanisms to accumulate and reuse past experiences[6, 7]. Such memory preserves observations, actions, reasoning steps, and solution trajectories, enabling agents to learn from prior successes and failures. It is therefore crucial for advanced capabilities such as personalization, experience-driven adaptation, and self-evolution[6–9]. However, designing memory systems that are both flexible to update and effective during reasoning remains a fundamental challenge.

Early memory systems for LLM-based agents mainly follow two paradigms: *explicit memory* and *parametric memory*. Explicit memory stores historical interactions as readable text and retrieves relevant content as additional context for future decisions[10, 11]. While this paradigm offers strong interpretability and adaptability, it suffers from high token overhead and limited representational capacity as injecting long trajectories increases context length and inference cost[12, 13]. Meanwhile,

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

plain text may fail to capture implicit procedural knowledge and latent decision patterns within complex experiences[14, 15]. In contrast, parametric memory internalizes past knowledge directly into model parameters through continuous training or fine-tuning[16–18], enabling memory to be more deeply integrated into the reasoning process and reducing prompt-time overhead. However, such memory is difficult to edit or update because stored experiences are entangled with model parameters and typically require additional optimization to modify[19, 20]. Moreover, domain-specific memory and general reasoning capability are coupled within the same parameters, so memory updates may unintentionally affect broader model behaviors, degrade general capabilities, and limit transferability to new scenarios[21, 22]. Therefore, an ideal agent memory system should not only efficiently store and represent historical experiences, but also remain flexible enough to allow the model to effectively use memory across domains.

Recently, *external latent memory* [23–25] has emerged as a promising direction. It adopts an encoder to represent rich trajectory-level information as compact latent vectors stored in an external memory bank, and learns to retrieve and utilize these latent representations during LLM reasoning. Latent memory methods aim to combine the best of both paradigms by learning powerful memory representations that can be more efficiently stored and retrieved than raw text, while keeping memory content outside the model so that it can be updated and transferred by rebuilding the memory bank. However, existing external latent memory approaches remain limited in two key aspects. First, their memory encoders are often optimized end-to-end with downstream task objectives, which can cause the encoder to overfit task-specific supervision signals rather than learn a generally reusable memory representation. Consequently, when deployed on new tasks or domains, the encoder may fail to produce meaningful latent memory representations that preserve transferable experience. Second, current memory-use mechanisms are still relatively weak for latent memories. Although latent representations can store richer and more compact experience than raw text, they are not directly interpretable by LLMs. As a result, the model may struggle to identify which latent memories are useful, determine how much information to extract from them, and decide when to invoke them during multi-step reasoning. These limitations suggest the need for a memory framework that decouples memory construction from downstream memory-use training while enabling LLMs to more effectively interpret and use latent memories.

To address these limitations, we propose MemPilot, a decoupled latent memory framework for memory-augmented reasoning. MemPilot separates memory construction from memory usage: it constructs an offline memory bank from trajectory-level latent representations while learning a lightweight mechanism to retrieve and use these memories during inference. Specifically, MemPilot first uses a memory encoder to compress reasoning trajectories into a latent memory bank. Once constructed, the memory entries are kept fixed during memory-use training. This prevents the subsequent retrieval-and-fusion module from changing the stored memory content to fit source-specific shortcuts, and encourages it to learn a reusable procedure for accessing fixed latent memories. Instead, with the base reasoner also kept frozen, the lightweight memory-use module is trained to retrieve relevant memories and interpret these fixed latent representations for the current reasoning process. It selects useful memory signals and integrates them with the model's hidden states using a gated fusion module. By decoupling the domain-specific memory content from this learned retrieval-and-fusion mechanism, MemPilot delivers dynamic, context-aware guidance during the reasoning steps, while naturally supporting a flexible form of cross-domain transfer. This decoupled nature enables a checkpoint trained on one task to be deployed to a new domain by simply rebuilding the offline memory bank with the same encoder, without any additional parameter updates.

Our contributions are summarized as follows:

- We introduce MemPilot, a decoupled latent external memory framework that stores domain-specific latent memories in an offline memory bank while learning a reusable mechanism to use fixed latent memories during reasoning.
- We design a retrieval and gated fusion mechanism that selects relevant compact trajectory memories and integrates useful memory signals into the model's hidden states, without appending long retrieved demonstrations to the prompt.
- We empirically validate MemPilot across diverse reasoning tasks, demonstrating strong in-domain performance and improved cross-domain transfer robustness. Ablations and visualizations further clarify the roles of retrieval, gating, and latent memory representations.

2 Related Work

Explicit Retrieval and Prompt-based Agent Memory. The most prevalent paradigm for equipping LLMs with memory relies on explicit textual retrieval[26–28]. In the context of LLM-based agents, explicit memory methods usually store historical interactions, demonstrations, or environment observations in external databases and inject retrieved content back into the working context[29, 30]. Frameworks such as *ReAct*[31] and recent *Agent Workflow Memory* methods[11] further highlight the importance of reusing past reasoning or action traces for complex multi-step tasks. While this transparent strategy requires no parameter updates and is easy to edit, it faces scalability and robustness bottlenecks. Repeatedly processing retrieved text increases the effective context length and inference cost. Moreover, explicit retrieval can suffer from "lexical traps", where retrieved demonstrations share surface similarities with the current query but require different reasoning paths, thereby distracting the model[14].

Parametric Memory. An alternative paradigm bypasses textual context expansion by representing memory in model parameters or continuous hidden representations. Parametric memory methods internalize information into model weights through continual fine-tuning or targeted model editing[32, 20], or use retriever-pretrained components such as *MLP Memory*[17] to store and access knowledge within neural layers. Although these approaches can integrate memory deeply into model computation, updating the stored content typically requires additional optimization and may couple domain-specific information with the model's general reasoning behavior.

Latent Memory. To retain external editability while avoiding long textual prompts, recent work explores latent memory, which stores continuous representations in external or semi-external memory structures[33]. For instance, *ICAE*[13] compresses long contexts into continuous tokens, *M+*[24] incorporates encoded knowledge vectors into hidden states for long-term memory, *MemGen*[34] studies generative latent memory for self-evolving agents, and *NextMem*[23] focuses on latent factual memory for reconstruction. Most of these works emphasize factual knowledge injection, context compression, or memory reconstruction. In contrast, our MemPilot focuses on procedural experience: it compresses reasoning trajectories into compact latent memory tokens and learns a reusable retrieval-and-fusion mechanism, enabling target-domain adaptation through memory-bank substitution rather than parameter updates.

3 Method

In this paper, we develop MemPilot, a decoupled latent memory interface that integrates external latent memories into the reasoning process of a frozen LLM. As illustrated in Figure 1, MemPilot consists of two phases. First, in the offline memory construction phase, historical reasoning trajectories are compressed into compact latent memory tokens and stored in an external memory bank. Second, in the memory-augmented reasoning phase, relevant memory entries are retrieved from the bank, read through a lightweight latent-memory interface, and fused into the hidden states of the frozen reasoner. This separation between memory content and the learned memory-use mechanism provides a natural protocol for target-domain adaptation: a new domain can be handled by rebuilding the offline memory bank while keeping the learned parameters fixed.

3.1 Problem Setup

We consider a family of reasoning tasks $\mathcal{D}$, where each task $d \in \mathcal{D}$ represents a domain such as code generation, mathematical reasoning, or question answering. Each task provides a training set $\mathcal{S}_d^{\mathrm{tr}}$ and a test set $\mathcal{S}_d^{\mathrm{te}}$. The training set serves as the available source of offline experience, containing queries, intermediate reasoning trajectories, and final outputs.

Our goal is to augment a pretrained, frozen LLM reasoner f_θ with an external latent memory module. We denote the offline memory encoder as E_ψ, which compresses training trajectories into latent memory tokens, and denote the memory-use mechanism as ϕ, which retrieves and fuses latent memories during reasoning. The generation reasoner f_θ remains frozen throughout training and inference; E_ψ is adapted only during offline memory construction and is frozen before memory-use training. For a given task d, we assume access to an offline latent memory bank $\mathcal{M}_d$ derived entirely from $\mathcal{S}_d^{\mathrm{tr}}$. To formalize the memory-augmented generation process, we decompose ϕ into two fundamental operations: retrieving relevant latent memories and fusing them into the active reasoning

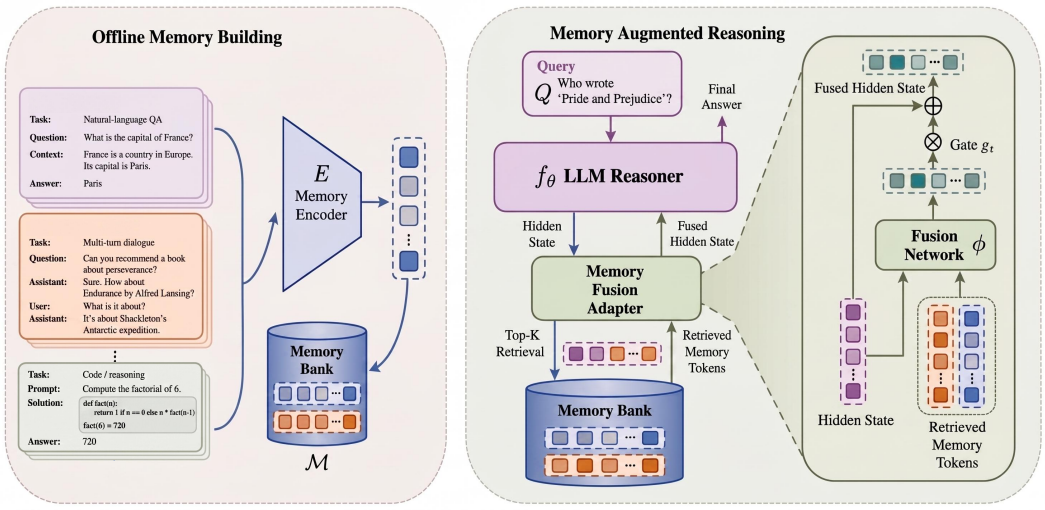


Figure 1: Overview of the proposed MemPilot framework.

process. For a test query $x \in \mathcal{S}_d^{\text{te}}$, the output is generated as:

$$H = f_\theta(x), \quad \tilde{Z} = \text{Ret}_\phi(H, \mathcal{M}_d), \quad H' = \text{Fuse}_\phi\left(H, \tilde{Z}\right), \quad \hat{y} = \text{Decode}_\theta(H'), \tag{1}$$

where Ret_ϕ denotes the memory retrieval function and Fuse_ϕ denotes the memory fusion module. Decode_θ denotes autoregressive decoding with the original output head of the frozen reasoner.

3.2 Offline Memory Construction

To ensure that the learned memory-use mechanism remains transferable, we structurally decouple memory construction from memory usage. The offline memory encoder E_ψ is a separate module from the generation reasoner f_θ. It compresses training trajectories into latent memory tokens for memory-bank construction, whereas f_θ performs autoregressive decoding and remains frozen.

Memory encoder alignment. The memory encoder can be instantiated either as a frozen base encoder or as an adapted encoder. In our main configuration, E_ψ is initialized from the same backbone architecture as f_θ and adapted with low-rank adapters attached only to the offline memory encoder. Before constructing the final memory bank, we perform an offline encoder-alignment stage that optimizes these encoder-side adapters together with the query/key projection heads used for memory retrieval. We use the contrastive retrieval objective in Eq. 8 to align query states with trajectory-level latent memories. Importantly, the encoder is not directly optimized with the downstream generation loss, which prevents the memory representations from co-adapting to the source-domain fusion module. After this stage, E_ψ is frozen and used only to construct the offline memory bank, which remains fixed during subsequent memory-use training. We also ablate the frozen base encoder in Section 4.3.

Latent memory bank construction. For each task d, we construct the offline latent memory bank $\mathcal{M}_d$ using the task-specific training set $\mathcal{S}_d^{\text{tr}}$. Rather than explicitly storing raw text, we compress the lengthy reasoning trajectory τ_i^d associated with each training sample into continuous representations. Specifically, we append one compression token and S memory tokens to the textual trajectory, and process the concatenated sequence with the memory encoder E_ψ. We discard the ordinary text-token hidden states and keep only the final hidden states at the S memory-token positions, obtaining a compact latent representation of the trajectory:

$$\mathbf{Z}_i^d = [\mathbf{z}_{i,1}^d; \ldots; \mathbf{z}_{i,S}^d] \in \mathbb{R}^{S \times h}, \tag{2}$$

where each $\mathbf{z}_{i,s}^d \in \mathbb{R}^h$ denotes a single memory token and S is the predefined number of tokens per sample. To enable efficient sample-level retrieval, we associate each latent memory sequence with

a retrieval key $\mathbf{k}_i^d$, obtained by pooling the memory encoder hidden states of the same trajectory. Consequently, the resulting memory bank is structured as a collection of key-value pairs, consisting of the retrieval keys and their corresponding compressed memory tokens:

$$\mathcal{M}_d = \{(\mathbf{k}_i^d, \mathbf{Z}_i^d)\}_{i=1}^{N_d}. \tag{3}$$

This design naturally forms a two-level memory structure: a coarse sample-level key ($\mathbf{k}_i^d$) for rapid candidate selection, and a fine-grained token-level memory set ($\mathbf{Z}_i^d$) for downstream reasoning fusion. By transforming raw training experiences into compact latent units, this offline bank avoids repeatedly processing long retrieved textual demonstrations during inference, thereby reducing the context-length overhead of text-based retrieval.

3.3 Latent Memory Retrieval and Gated Fusion

During memory-augmented reasoning, we apply the memory interface after the final transformer block and before the language-modeling head. Retrieval is performed once using the query representation at the beginning of generation, and the retrieved latent memory tokens are reused during autoregressive decoding. A lightweight cross-attention reader then performs token-level memory reading over the retrieved tokens, and the resulting memory signal is fused into the reasoner's hidden states through a gated residual connection.

Candidate memory retrieval. A straightforward formulation is to compute global cross-attention between the active hidden states and all latent memory tokens in the memory bank. However, dense attention over a large fine-grained memory bank is computationally expensive and can introduce noise from irrelevant memories. To make memory access both efficient and robust, we adopt a coarse-to-fine pipeline: MemPilot first retrieves top-K sample-level candidate entries and then performs token-level memory reading over the selected latent tokens. Given an input query x, the frozen reasoner produces a hidden sequence $\mathbf{H} \in \mathbb{R}^{L\times h}$ from its final transformer block, before the LM head computes the next-token distribution. We derive a dense query representation $\mathbf{q} \in \mathbb{R}^h$ by pooling the query-side hidden states in $\mathbf{H}$ before autoregressive generation begins. In the discrete sample-level retrieval stage, we score $\mathbf{q}$ against the sample-level keys in $\mathcal{M}_d$:

$$\alpha_i = \langle \mathbf{q}W_Q^{\text{ret}}, \mathbf{k}_i^d W_K^{\text{ret}} \rangle, \tag{4}$$

where W_Q^{ret} and W_K^{ret} are retrieval projection matrices. Based on the alignment scores α_i, we select the top-K candidate samples, where K denotes the number of retrieved sample-level entries.

Memory reading and gated fusion. With the relevant historical contexts retrieved, the next critical step is to utilize this latent knowledge to assist the active reasoning process. We formulate this stage as a fine-grained interaction mechanism, where the model selectively extracts related information from the retrieved latent representations to guide its current inference step. Specifically, we flatten the selected candidates into a compact memory matrix $\widetilde{\mathbf{Z}} = [\mathbf{Z}_{i_1}^d; \ldots; \mathbf{Z}_{i_K}^d] \in \mathbb{R}^{KS\times h}$, enabling the active hidden states $\mathbf{H}$ to act as queries that attend over the memory matrix $\widetilde{\mathbf{Z}}$. In this process, the attention mechanism dynamically assigns higher weights to memory tokens that share greater representational similarity with the current reasoning context. We compute this attended memory representation $\mathbf{C}$ as:

$$\mathbf{C} = \text{CrossAttn}(\mathbf{H}W_Q^{\text{attn}}, \widetilde{\mathbf{Z}}W_K^{\text{attn}}, \widetilde{\mathbf{Z}}W_V^{\text{attn}}). \tag{5}$$

The resulting readout $\mathbf{C}$ thus functions as a contextually weighted synthesis of historical knowledge. By dynamically aligning past experiences with the active context, this representation enriches the model with targeted procedural hints, effectively providing dynamic latent guidance to assist the active reasoning process.

However, not every inference step requires external augmentation. In scenarios where the current reasoning state is entirely unrelated to the retrieved memories, forcibly integrating the memory representation $\mathbf{C}$ would introduce noise and perturb the base reasoner's inherent logic. To prevent this forced injection, we introduce a learned gating mechanism $\mathbf{g} = \sigma(\mathbf{H}W_g)$. This gate acts as a dynamic filter capable of suppressing the external signal when the memory is deemed unhelpful, modulating the fused memory before it enters the model's residual stream:

$$\mathbf{H}' = \mathbf{H} + \delta \cdot (\mathbf{g} \odot \mathbf{C})W_O, \tag{6}$$

where $\delta \in (0, 1)$ is a scaling factor. The fused hidden sequence $\mathbf{H}'$ is then passed to the original LM head to compute the next-token distribution. By combining this gated control with a strict residual design, MemPilot ensures that the external bank serves as an adaptive corrective signal, injecting knowledge only when necessary, without compromising the base model's computational stability.

3.4 Training Objective

The goal of training is not to absorb source-domain knowledge into the generation reasoner, but to learn a reusable interface for using fixed external latent memories. After the offline encoder-alignment stage, we freeze E_ψ and construct the memory bank $\mathcal{M}_d$ from $\mathcal{S}_d^{\mathrm{tr}}$. During memory-use training, both the generation reasoner f_θ and the memory bank $\mathcal{M}_d$ are fixed, and we optimize only the memory-use mechanism ϕ. Concretely, ϕ learns three domain-agnostic operations: selecting relevant memories, reading their latent memory tokens, and injecting useful memory signals into the frozen reasoner's hidden states. Given a training example (x_i, y_i, τ_i), the memory-conditioned generation objective is

$$\mathcal{L}_{\mathrm{gen}} = -\sum_t \log p_{\theta,\phi}\left(y_{i,t} \mid x_i, y_{i,<t}, \tilde{Z}_i\right), \tag{7}$$

where the distribution $p_{\theta,\phi}$ is computed by applying the memory fusion module to the hidden states of the frozen reasoner before decoding. This objective encourages the memory-use mechanism to use retrieved latent memories as auxiliary guidance for the frozen reasoner, thereby improving the final reasoning output.

To make the external bank searchable, we use a contrastive retrieval objective. For each query x_i, the positive set $\mathcal{P}_i$ contains the memory entry constructed from its corresponding reasoning trajectory, while $\mathcal{C}_i$ contains both positive and negative candidates sampled from the bank. With retrieval score α_{ij}, we define

$$\mathcal{L}_{\mathrm{ret}} = -\log \frac{\sum_{j\in\mathcal{P}_i} \exp(\alpha_{ij}/\epsilon)}{\sum_{j\in\mathcal{C}_i} \exp(\alpha_{ij}/\epsilon)}, \tag{8}$$

where ϵ is a temperature factor. The retrieval loss is used in both stages, but with different trainable components. During encoder alignment, it updates the encoder-side adapters and retrieval projections; during memory-use training, with E_ψ and the memory bank fixed, it continues to optimize the retrieval component within ϕ. The main training objective is therefore

$$\mathcal{L} = \mathcal{L}_{\mathrm{gen}} + \lambda_{\mathrm{ret}}\mathcal{L}_{\mathrm{ret}}. \tag{9}$$

This objective directly supports the decoupling principle of our method. The domain-specific information appears only through the external bank $\mathcal{M}_d$, whereas the learned parameters ϕ are optimized to perform a general procedure: retrieve relevant latent memory tokens and integrate them into the ongoing reasoning process through gated fusion. For target-domain d_t deployment, we keep f_θ, E_ψ, and $\phi^\star$ fixed and rebuild only the memory bank $\mathcal{M}_{d_t}$ from $\mathcal{S}_{d_t}^{\mathrm{tr}}$ using the fixed memory encoder. Thus, adaptation is realized through memory-bank substitution rather than parameter fine-tuning. Further implementation details, training settings, and hyperparameter configurations are provided in Appendix A.1.

4 Experiments

4.1 Experimental Settings

Evaluation and Benchmarks. We evaluate MemPilot across diverse reasoning tasks to assess both in-domain performance and cross-domain generalization. Our evaluation covers five datasets from three primary domains: **mathematical reasoning**, including GSM8K [35] and MATH [36]; **question answering**, including HotpotQA [37] and TriviaQA [38]; and **executable code synthesis**, using KodCode [39]. Together, these benchmarks cover factual retrieval, multi-step reasoning, and executable problem solving, providing a broad testbed for evaluating memory-augmented reasoning.

Baselines. To demonstrate the superiority of our approach across different memory paradigms, we compare our proposed method MemPilot against strong baselines categorized into three groups: (I) **Standard and fine-tuning methods**: Vanilla base model, supervised fine-tuning (SFT), and Agent-FLAN [40]; (II) **Retrieval-based memory frameworks**, where historical trajectories are appended

Table 1: Main in-domain results. Best scores are in **bold**, and second-best scores are underlined.

Method	QA		Coding	Math		Overall
	HotpotQA	TriviaQA	KodCode	GSM8K	MATH	
Qwen2.5-1.5B-Instruct						
Vanilla	47.39	32.10	24.55	56.79	36.63	39.49
SFT	50.16	63.84	55.83	54.83	38.84	52.70
Agent-FLAN	52.12	64.28	56.21	53.02	29.82	51.09
ExpeL	55.81	25.20	31.15	45.12	38.12	39.08
MemoryBank	55.39	38.14	37.93	47.88	30.47	41.96
SoftCoT	55.32	62.22	55.13	54.50	38.55	53.14
Co-processor	53.22	64.78	56.65	57.12	37.40	53.83
MemGen	56.01	65.02	58.16	58.15	**47.12**	56.89
MemPilot	**59.29**	**66.18**	**58.55**	**59.97**	46.32	**58.06**
Qwen3-8B						
Vanilla	58.39	52.18	49.10	89.48	79.82	65.79
SFT	62.12	74.55	64.75	90.76	81.35	74.71
Agent-FLAN	59.15	70.32	62.99	87.60	80.05	72.02
ExpeL	55.76	65.54	57.20	86.20	77.40	68.42
MemoryBank	61.14	60.56	56.39	90.35	80.35	69.76
SoftCoT	63.53	59.42	63.28	86.30	76.23	69.75
Co-processor	65.12	61.42	64.90	76.23	79.20	69.37
MemGen	63.15	**77.22**	66.15	91.25	**83.30**	76.21
MemPilot	**66.83**	77.09	**66.82**	**91.68**	82.86	**77.06**

as textual demonstrations, represented by ExpeL [41] and MemoryBank [10]; and (III) **Latent memory and computation frameworks**, which leverage latent tokens for experiential knowledge, including SoftCoT [15], Co-processor [42], and MemGen [34], a recent framework that dynamically synthesizes latent memory tokens to interleave with the reasoning process.

Implementation Details. We select LLM backbones of different sizes, specifically Qwen2.5-1.5B-Instruct and Qwen3-8B. Unless otherwise specified, the number of retrieved memory entries is set to $K = 4$, and the number of latent memory tokens per entry is set to $S = 8$. Further training details, evaluation metrics and hyperparameter configurations are provided in Appendix A.1.

4.2 Main Results

In-Domain Evaluation. Table 1 reports the in-domain results on both Qwen2.5-1.5B-Instruct and Qwen3-8B. Overall, MemPilot achieves strong and consistent performance across reasoning, coding, and knowledge-intensive tasks. Compared with latent or parametric memory baselines, MemPilot achieves the best overall performance on both Qwen2.5-1.5B-Instruct and Qwen3-8B. While MemGen marginally leads on a few individual datasets, MemPilot provides stronger overall results and a more flexible memory design. In particular, MemGen couples memory behavior more closely with trainable parameters, whereas MemPilot keeps domain-specific experience externalized in a replaceable memory bank and only learns a lightweight retrieval-and-fusion mechanism for using it. This structural decoupling is crucial for flexible memory substitution and cross-domain deployment, which we evaluate next. Beyond accuracy, we provide an efficiency analysis in Section 4.4.

Cross-Domain Transferability. To evaluate whether the memory-use mechanism can generalize beyond its source domain, we conduct cross-domain transfer experiments where each model is optimized on a single source dataset and directly evaluated on held-out target domains. For MemPilot, target-domain adaptation is performed by rebuilding only the offline memory bank, while keeping both the generation reasoner and the learned retrieval-and-fusion module unchanged. To control for access to target-domain training trajectories, we additionally include ExpeL and MemoryBank as source-agnostic target text-memory baselines, which rebuild explicit textual memory from the target training split without parameter updates. As shown in Table 2, implicit parametric memory exhibits clear signs of domain coupling under this setting. For example, when trained on HotpotQA,

Table 2: Cross-domain transfer results. In-domain results are shown in gray to distinguish them from out-of-domain transfer results.

Source Train	Method	Target Evaluation				
		KodCode	GSM8K	MATH	HotpotQA	Overall
–	Vanilla Base	24.55	56.79	36.63	47.39	41.34
–	ExpeL	31.15	45.12	38.12	55.81	42.55
–	MemoryBank	37.93	47.88	30.47	55.39	42.92
KodCode	SFT	55.83	8.12	3.20	35.67	25.71
	MemGen	58.16	33.87	22.22	45.72	39.99
	MemPilot (Ours)	58.55	**63.01**	**40.92**	**50.17**	**53.16**
GSM8K	SFT	20.56	54.83	10.76	45.26	32.85
	MemGen	21.34	58.15	26.40	49.94	38.96
	MemPilot (Ours)	**27.07**	59.97	**36.82**	**50.56**	**43.61**
MATH	SFT	26.15	50.86	38.84	43.14	39.75
	MemGen	23.12	53.54	47.12	48.13	42.98
	MemPilot (Ours)	**29.30**	**59.22**	46.32	**52.56**	**46.85**
HotpotQA	SFT	15.61	21.60	10.45	50.16	24.46
	MemGen	3.12	5.34	2.14	56.01	16.65
	MemPilot (Ours)	**27.65**	**62.12**	**28.16**	59.29	**44.31**

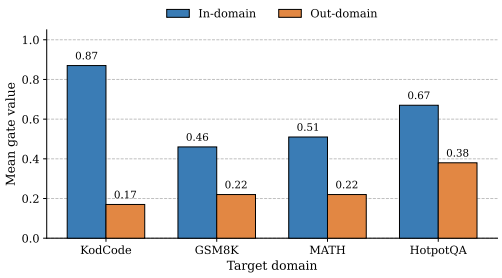


Figure 2: Mean gate value across domains, showing adaptive suppression in OOD scenarios.

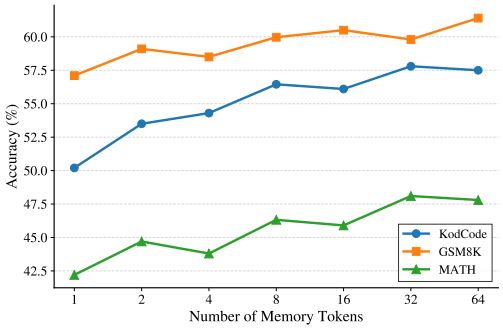


Figure 3: Impact of Memory Sequence Length.

MemGen drops to 3.12% on KodCode and 2.14% on MATH, far below the frozen Vanilla baseline. This suggests that parameterized memory can entangle source-domain regularities with the model's reasoning behavior, leading to negative transfer when the target reasoning pattern differs substantially.

In contrast, MemPilot demonstrates substantially stronger transfer robustness. When trained on KodCode and evaluated with target-domain memory banks, it achieves 63.01% on GSM8K and 40.92% on MATH, outperforming the Vanilla baseline without any target-domain parameter updates. Similar trends are observed across other source domains, where MemPilot generally preserves or improves the base model's performance more reliably than MemGen. Compared with the target text-memory baselines, MemPilot also achieves stronger overall transfer performance, suggesting that target-domain trajectories alone are not sufficient. Since these memories are full reasoning trajectories rather than clean evidence documents, explicit text retrieval can introduce irrelevant or mismatched intermediate steps into the prompt. Across the 12 off-diagonal pairs, MemPilot achieves an average score of 43.96 with target-domain memory banks. With the same access to target trajectories, ExpeL and MemoryBank average 42.55 and 42.92, respectively. The source-trained SFT and MemGen baselines receive no target trajectories in this comparison. These results support the benefit of structural decoupling: domain-specific experience is stored externally in the memory bank, while the learned module mainly captures a reusable procedure for retrieving and fusing latent memories. We also observe that transfer gains depend on the affinity between source and target reasoning modalities. For instance, KodCode transfers well to mathematical reasoning, likely because algorithmic code synthesis and formal problem solving share procedural structures. This suggests that MemPilot primarily transfers a reusable memory-use procedure, rather than relying on source-domain content memorization; when reasoning modalities differ more substantially, the gains become more moderate but remain less prone to severe negative transfer than parameter-coupled baselines.

4.3 Ablation Study and Analysis

Effect of the Retrieval and Gating Modules. To systematically investigate the internal mechanics underlying MemPilot's robust cross-domain transferability, we jointly analyze the inference-time component ablation (Table 3) and the gate activation patterns (Figure 2). As shown in Table 3, replacing the learned retrieval model with random memory sampling (*Random Ret.*) significantly degrades performance across evaluations (e.g., -13.43% on KodCode). This suggests that irrelevant or structurally mismatched memories can substantially impair reasoning, rather than serving as useful auxiliary context. The learned retrieval module is therefore essential for selecting memories that are compatible with the current reasoning state before they are passed to the fusion module. Beyond retrieval relevance, the gating module is also crucial for controlling how much retrieved memory should influence the current reasoning process. As shown in Table 3, removing the gate (*w/o Gate*) substantially hurts out-of-domain performance and falls below the frozen Vanilla Base on MATH. This suggests that blindly fusing retrieved memories can introduce harmful noise when the memory signals are not well aligned with the current reasoning modality. As shown in Figure 2, the mean gate value remains high in in-domain settings, reaching 0.87 on KodCode, but decreases to 0.17 in the most mismatched out-of-domain setting. This indicates that the gating mechanism does not simply inject memory by default; instead, it regulates memory influence according to the compatibility between the retrieved memories and the current reasoning state. As a result, MemPilot can benefit from useful latent guidance while reducing interference from mismatched memory signals, helping preserve the frozen reasoner's intrinsic reasoning capability across domains.

Table 3: Inference-time component ablations. Numbers in parentheses denote degradation relative to the full MemPilot.

Method	In-Domain	Out-of-Domain		
	KodCode	GSM8K	MATH	OOD Avg.
Vanilla Base	24.55	56.79	36.63	46.71
MemPilot (Random Ret.)	45.12 (-13.43)	54.21 (-8.80)	35.10 (-5.82)	44.66 (-7.31)
MemPilot (w/o Gate)	54.77 (-3.78)	52.34 (-10.67)	31.76 (-9.16)	42.05 (-9.92)
MemPilot	**58.55**	**63.01**	**40.92**	**51.97**

Impact of Memory Sequence Length. The number of latent memory tokens (S) controls the memory sequence length and representational capacity of the offline trajectories. To investigate its impact, we vary S across three distinct in-domain settings. As shown in Figure 3, overall performance generally improves as the number of memory tokens increases. When the sequence is overly constrained, performance remains sub-optimal due to an information bottleneck. As the number of latent memory tokens increases up to 8, the models exhibit a substantive upward trend in accuracy, indicating that the information bottleneck is progressively alleviated. In this paper, we select $S = 8$ to achieve a balance between representational completeness and inference efficiency.

Table 4: Encoder ablation study.

Configuration	KodCode		GSM8K		MATH	
	Score	Δ	Score	Δ	Score	Δ
Frozen reasoner (No Mem.)	24.5	–	56.7	–	36.6	–
MemPilot w/ frozen base encoder	53.3	+28.8	59.3	+2.6	43.2	+6.6
MemPilot w/ LoRA memory encoder	**58.6**	**+34.1**	**60.0**	**+3.3**	**46.3**	**+9.7**

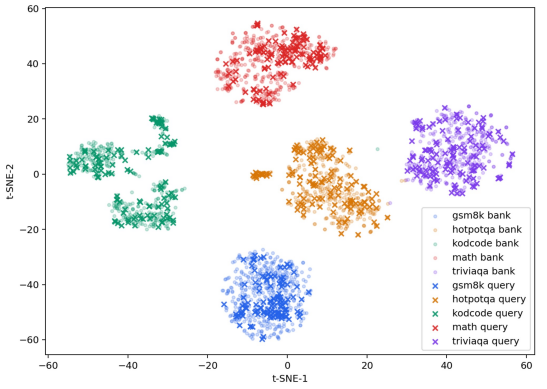


Figure 4: t-SNE visualization of latent memory representations extracted by the frozen Base encoder, demonstrating natural domain clustering.

Ablation on Memory Encoder. To understand the impact of memory representation quality on our framework, we conduct an ablation study comparing the frozen base encoder against a LoRA-adapted memory encoder, while keeping the generation reasoner frozen. Table 4 and Figure 4 detail the performance gains and the underlying representation distributions. Remarkably, MemPilot achieves substantial performance improvements even when relying solely on the frozen base encoder. To elucidate this phenomenon, we visualize the latent trajectories extracted by the base encoder using t-SNE (Figure 4). The visualization reveals that the untuned representations naturally self-organize into distinct, highly separable clusters corresponding to their respective domains. This indicates that the base LLM already possesses a strong inherent capacity to encode modality-specific structural

semantics. Consequently, the fusion module can effectively leverage these naturally clustered latent spaces to retrieve relevant reasoning signals without requiring explicit encoder fine-tuning.

While the base encoder provides a solid foundation, incorporating a LoRA-adapted memory encoder yields additional gains. As shown in Table 4, replacing the frozen base encoder with the LoRA memory encoder further improves MemPilot across all evaluated domains. This suggests that although the base encoder's natural clustering is sufficient for broad domain differentiation, LoRA adaptation further refines the latent memory space by aligning trajectory representations more tightly with the retrieval mechanism's optimization objective.

4.4 Efficiency Analysis

Table 5 reports the inference-time cost on KodCode. RAG retrieves four raw training trajectories from the same pool used by MemPilot. Compared with RAG, MemPilot removes the need to append retrieved textual demonstrations, reducing the prompt-side token overhead while achieving a higher score. Although the vanilla Base LM does not use retrieval, it generates much longer outputs and has higher measured latency. SFT also produces shorter outputs than the base LM and runs slightly faster than MemPilot (1.05 vs. 1.12 s/example), while MemPilot achieves a higher score (58.55 vs. 55.83). Given training trajectories, MemPilot adapts to a new domain by encoding a bank in 45 s, compared with about 2.1 h for SFT and 2.5 h for MemGen retraining. Further cost and storage details are reported in Appendix A.2.

Table 5: Inference-time cost comparison on KodCode.

Method	Score	Input Tok.	Ret. Text Tok.	Output Tok.	s/ex.
Base LM	24.55	256.62	0	511.36	3.51
RAG	40.15	256.62	+2,024.60	511.40	5.87
SFT (LoRA)	55.83	256.62	0	146.52	**1.05**
MemGen	58.16	256.62	0	122.60	3.34
MemPilot	**58.55**	256.62	0	102.36	1.12

5 Conclusion

In this work, we introduce MemPilot, a decoupled latent memory framework that bridges the flexibility of explicit textual retrieval and the reasoning depth of implicit parametric memory. By externalizing domain-specific experience into compact latent memory banks and learning a reusable retrieval-and-fusion mechanism, MemPilot can dynamically guide the hidden reasoning states of a frozen LLM without appending long textual demonstrations or updating the base model. Experiments across diverse reasoning domains show that MemPilot improves in-domain performance while supporting cross-domain transfer through simple offline bank substitution. Further analysis demonstrates that the retrieval and adaptive gating components are both important for selecting useful latent memories and suppressing harmful out-of-domain signals. Overall, MemPilot offers a flexible and efficient path toward memory-augmented reasoning, where memory content remains external and replaceable, while the learned memory-use mechanism can be reused across domains.

Acknowledgments and Disclosure of Funding

Yi Chang was supported by the Ministry of Science and Technology of China (No. 2023YFF0905400), the National Natural Science Foundation of China (No. U2341229), and the New Cornerstone Science Foundation through the XPLORER PRIZE. We extend our special thanks to Mr. Chenhan Wang from Hyper.AI for generously providing the computing resources essential to this research.

The authors have no competing interests to declare.

References

[1] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

[2] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.

[3] Shuo Ren, Can Xie, Pu Jian, Zhenjiang Ren, Chunlin Leng, and Jiajun Zhang. Towards scientific intelligence: A survey of llm-based scientific agents. *arXiv preprint arXiv:2503.24047*, 2025.

[4] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459*, 2024.

[5] Wenlin Zhang, Xiaopeng Li, Yingyi Zhang, Pengyue Jia, Yichao Wang, Huifeng Guo, Yong Liu, and Xiangyu Zhao. Deep research: A survey of autonomous research agents. *arXiv preprint arXiv:2508.12752*, 2025.

[6] 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.

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

[8] Guibin Zhang, Muxin Fu, Kun Wang, Guancheng Wan, Miao Yu, and Shuicheng YAN. G-memory: Tracing hierarchical memory for multi-agent systems. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL `https://openreview.net/forum?id=mmIAp3cVS0`.

[9] Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, et al. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*, 2025.

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

[11] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning*, pages 63897–63911. PMLR, 2025.

[12] In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In P. Gibbons, G. Pekhimenko, and C. De Sa, editors, *Proceedings of Machine Learning and Systems*, volume 6, pages 325–338, 2024. URL `https://proceedings.mlsys.org/paper_files/paper/2024/file/a66caa1703fe34705a4368c3014c1966-Paper-Conference.pdf`.

[13] Tao Ge, Hu Jing, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=uREj4ZuGJE`.

[14] Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pages 31210–31227. PMLR, 2023.

[15] Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. Softcot: Soft chain-of-thought for efficient reasoning with llms. *arXiv preprint arXiv:2502.12134*, 2025.

[16] Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh R N, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phil L Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. Retroformer: Retrospective large language agents with policy gradient optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KOZu91CzbK.

[17] Rubin Wei, Jiaqi Cao, Jiarui Wang, Jushi Kai, Qipeng Guo, Bowen Zhou, and Zhouhan Lin. MLP memory: A retriever-pretrained memory for large language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=1SMdxRtLBp.

[18] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.181. URL https://aclanthology.org/2024.findings-acl.181/.

[19] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=0DcZxeWfOPt.

[20] Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=MkbcAHIYgyS.

[21] Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 2025.

[22] Akshat Gupta, Anurag Rao, and Gopala Anumanchipalli. Model editing at scale leads to gradual and catastrophic forgetting. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15202–15232, 2024.

[23] Zeyu Zhang, Rui Li, Xiaoyan Zhao, Yang Zhang, Wenjie Wang, Xu Chen, and Tat-Seng Chua. Nextmem: Towards latent factual memory for llm-based agents. *arXiv preprint arXiv:2603.15634*, 2026.

[24] Yu Wang, Dmitry Krotov, Yuanzhe Hu, Yifan Gao, Wangchunshu Zhou, Julian McAuley, Dan Gutfreund, Rogerio Feris, and Zexue He. M+: Extending memoryLLM with scalable long-term memory. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=OcqbkROe8J.

[25] Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. Memoryllm: towards self-updatable large language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[26] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[27] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR, 2022.

[28] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

[29] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

[30] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[31] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

[32] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022.

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

[34] Guibin Zhang, Muxin Fu, and Shuicheng YAN. Memgen: Weaving generative latent memory for self-evolving agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=vI56m4Iu4e.

[35] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[36] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Thirty-Fifth Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

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

[38] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, 2017.

[39] Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025.

[40] Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-flan: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

[41] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642, 2024.

[42] Luyang Liu, Jonas Pfeiffer, Jiaxing Wu, Jun Xie, and Arthur Szlam. Deliberation in latent space via differentiable cache augmentation. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=IaUJl5RCOu.

A Technical appendices and supplementary material

A.1 Experiment Settings

Implementation Details and Metrics. For evaluation, we employ task-specific metrics. For GSM8K and MATH, we report exact-answer accuracy after extracting and normalizing the final answer. For KodCode, we report execution accuracy, measured by the fraction of test cases passed. For HotpotQA, we evaluate on the easy subset with all provided context paragraphs available to the model. We report answer-level Exact Match (EM) and token-level F1 after normalization; unless otherwise specified, the scalar score corresponds to answer EM. For TriviaQA, we report alias-match accuracy: a prediction is counted as correct if the final answer span contains any normalized gold answer alias.

Training details. We train the memory-use modules with AdamW, a cosine learning-rate schedule, bf16 precision, and a warmup ratio of 0.1. Unless otherwise stated, each stage is trained for one epoch. Evaluation uses greedy decoding with the target-domain memory bank rebuilt offline and no target-domain parameter updates.

Parameterization and training stages. We distinguish the *generation reasoner* from the *memory encoder*. The generation reasoner f_θ is the LLM used for autoregressive decoding, and its dense backbone parameters remain frozen throughout all experiments. The memory encoder E_ψ is a separate offline compression module used to convert training trajectories into latent memory tokens. Although E_ψ is initialized from the same backbone architecture, the LoRA adapters used in our implementation are attached only to the memory encoder for representation alignment. These encoder-side adapters are not inserted into the generation reasoner during decoding.

Training proceeds in two stages. First, during offline memory encoder training, we optimize only memory-related parameters, including the encoder-side LoRA adapters and the retrieval/interface projections needed to align query states with the latent memory space. After this stage, the memory encoder is frozen and used to construct the offline memory bank. Second, during memory-use training, the generation reasoner remains frozen, and we train only the lightweight retrieval-and-fusion module, including the latent cross-attention reader, gating module, output projection, and residual scaling parameter. During target-domain transfer, all learned parameters are fixed; adaptation is performed only by rebuilding the target-domain memory bank with the fixed memory encoder.

Offline bank construction. For each training trajectory, we append a learned compression token and S learned memory tokens, encode the resulting sequence with the fixed memory encoder E_ψ, and store the hidden states at the memory-token positions. A sample-level retrieval key is stored together with the corresponding memory-token set. The resulting memory bank is frozen after construction and serves as an external latent memory source during evaluation.

Inference-time memory use. Given a query, MemPilot first pools the generation reasoner's hidden states into a query vector. It retrieves the top-K sample-level memory entries, materializes the selected entries into latent memory tokens, and fuses their memory tokens into the active hidden states through gated cross-attention. The language-model head then decodes from the memory-conditioned hidden states. During this process, the generation reasoner, memory encoder, retrieval module, and fusion module are all fixed; no target-domain parameter updates are performed.

A.2 Additional Cost Details

Given training trajectories, index construction for trajectory RAG takes 2.9 s, while MemPilot bank encoding takes 45 s. SFT, MemGen, and MemPilot have 9.23M, 23.20M, and 11.64M trainable parameters, respectively. Using the same HotpotQA trajectory pool, the text RAG store occupies 46.90 MB, compared with 23.14 MB for the MemPilot latent bank.

A.3 Compute Resources

All experiments were performed on two NVIDIA RTX PRO 6000 Blackwell GPUs, each with 96GB GPU memory. The reported experiments mainly involve offline memory-bank construction, memory-use training, and inference evaluation.

A.4 Limitations

Although MemPilot demonstrates strong results across question answering, code generation, and mathematical reasoning, our evaluation is still limited to a finite set of benchmarks and two LLM backbones. Constructing a target-domain memory bank requires target-domain training trajectories. When only the source-domain memory bank is used, transfer gains are modest and depend on the source domain. Future work could further examine its behavior on larger-scale models, longer-horizon interactive agents, and substantially larger memory banks.

A.5 Broader Impact

MemPilot is a general memory-augmented reasoning framework that aims to make LLM memory more modular, updateable, and transferable. By externalizing domain-specific experience into replaceable memory banks, it may reduce the need for repeated parameter fine-tuning when adapting models to new domains. As with other memory-augmented LLM systems, practical deployments should ensure that memory banks are constructed from appropriate and authorized data. This work does not introduce new datasets involving sensitive personal information, nor does it release a new pretrained model.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction accurately reflect the paper's contributions and scope

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses limitations in Appendix A.4.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The method section specifies the architecture, retrieval and fusion modules, and training objectives, Section 4.1 and Appendix A.1 describe the benchmarks, baselines, backbones, metrics, trainable components, memory-bank construction, and transfer protocol.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The code is being prepared for public release and is not yet publicly available.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Section 4.1 and Appendix A.1 describe the experimental settings.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: We do not report error bars, but the results show consistent trends across datasets and ablations.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report the compute resources used for the experiments in Appendix A.3, including the GPU type, number of GPUs, GPU memory, and the main experimental stages involving memory-bank construction, memory-use training, and inference evaluation.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses public benchmarks and pretrained language models, does not involve human subjects or sensitive personal data collection, and is conducted in accordance with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix A.5 discusses the broader impact of MemPilot as a modular and transferable memory-augmented reasoning framework, and notes that practical deployments should construct memory banks from appropriate and authorized data.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a new high-risk pretrained model, image generator, scraped dataset, or other asset requiring special release safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper properly cites the original sources for all environments and baseline methods used in the experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not introduce or release a new dataset, benchmark, or pretrained model as a research asset.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The paper uses LLMs as the frozen base reasoners in both the proposed method and experiments. The evaluated backbones, including Qwen2.5-1.5B-Instruct and Qwen3-8B, are explicitly specified in the experimental setup.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.