

# MemSearcher: Iterative Memory Integration for Search Agent via End-to-End Reinforcement Learning

Qianhao Yuan<sup>1,2</sup>, Jie Lou<sup>3</sup>, Zichao Li<sup>1,2</sup>, Jiawei Chen<sup>1,2</sup>,  
Yaojie Lu<sup>1\*</sup>, Hongyu Lin<sup>1</sup>, Le Sun<sup>1</sup>, Debing Zhang<sup>3</sup>, Xianpei Han<sup>1</sup>

<sup>1</sup>Chinese Information Processing Laboratory, Institute of Software,

Chinese Academy of Sciences, Beijing, China

<sup>2</sup>University of Chinese Academy of Sciences, Beijing, China

<sup>3</sup>Xiaohongshu Inc

{yuanqianhao2024, lizichao2022, chenjiawei2024}@iscas.ac.cn

{luyaojie, hongyu, sunle, xianpei}@iscas.ac.cn loujie0822@gmail.com

## Abstract

LLM-based search agents often concatenate the full interaction history into the context, producing long and noisy inputs, and increasing compute cost and GPU memory overhead. To address this issue, we propose MemSearcher, an agent framework that maintains a compact memory during multi-turn interactions, retaining only question-relevant information and thereby keeping the context length stable across turns. Training MemSearcher is challenging because each trajectory spans multiple turns under different LLM contexts, making each turn an independent optimization target in reinforcement learning. We introduce multi-context GRPO, which propagates trajectory-level advantages to all turns for end-to-end optimization. Experiments demonstrate that MemSearcher outperforms strong history-concatenation (ReAct-style) baselines on a range of public datasets while maintaining nearly constant token counts across multi-turn interactions. The code and models will be publicly available at <https://github.com/icpicas/MemSearcher>.

## 1 Introduction

Recently, Large Language Model (LLM)-based search agents have advanced quickly, delivering substantial improvements on knowledge-acquisition tasks (Jin et al., 2025; Chen et al., 2025b; Li et al., 2025). Unlike traditional Retrieval-Augmented Generation (RAG) (Lewis et al., 2020; Yue et al., 2024; Xiong et al., 2025), search agents treat search engines as external tools, and autonomously determine when and how to invoke them. This process typically requires multiple turns of interaction between agents and the environment.

A representative paradigm for modeling such multi-turn interactions is ReAct (Yao et al., 2023).

\*Corresponding author.

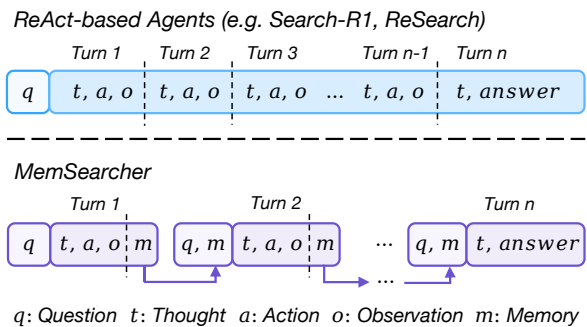


Figure 1: Comparison between ReAct and MemSearcher. ReAct continuously appends all interaction history, including thought  $t$ , action  $a$  and observation  $o$  at each turn. MemSearcher iteratively updates a compact memory  $m$  that retains only essential information.

As shown in Figure 1 (Top), a ReAct trajectory is formulated as a multi-turn conversation, where the entire interaction history is incorporated into the context of the backbone LLM. Although straightforward, this design leads to a fundamental bottleneck: as the number of interaction turns increases, the LLM context grows linearly by appending all thoughts, actions, and observations. In the setting of search agents, this results in long and noisy contexts dominated by retrieved passages, many of which are irrelevant to the user’s question. Such prolonged contexts not only strain the LLM’s ability to effectively utilize salient information (Liu et al., 2023; Wu et al., 2024), but also incur substantial computational and GPU memory overhead.

In this paper, we introduce MemSearcher, an agent framework that maintains a compact, iteratively updated memory throughout multi-turn interactions, preserving only the information essential for addressing the user’s question. At each turn, MemSearcher provides the backbone LLM with two succinct inputs, the user’s question and the memory, rather than the entire, ever-growing interaction history. The LLM generates a reasoning

trace (thought) and performs an action based on it. After the response to the action is returned by the environment, the LLM functions as a memory manager to integrate the memory based on the previous memory and the current interaction. Since the number of tokens in the memory is restricted by a predefined maximum length, this design keeps LLM contexts stable while preserving salient information throughout multi-turn interactions.

Since current LLMs have not been optimized under the MemSearcher framework, they are not yet capable of mastering it. We employ Reinforcement Learning (RL) (Sutton et al., 1998) to train MemSearcher agents, which enables the optimization of models without annotated trajectories. Among RL algorithms, Group Relative Policy Optimization (GRPO) (Shao et al., 2024) has emerged as a widely adopted method, as it improves LLM abilities while optimizing the GPU memory usage of Proximal Policy Optimization (PPO) (Schulman et al., 2017). In MemSearcher, a trajectory consists of multiple turns under different contexts, making each turn an independent optimization targets. To enable end-to-end GRPO training of MemSearcher agents, we introduce multi-context GRPO. It propagates trajectory-level advantages to each turn, and then optimizes every turn independently, enabling a stable training for MemSearcher agents.

We conduct extensive experiment across seven public knowledge-acquisition datasets. MemSearcher demonstrates superior performance to ReAct-based methods. Compared to ReAct, which exhibits a linear increase in token numbers with interaction turns, MemSearcher maintains nearly constant token counts within contexts. Notably, MemSearcher supports multi-turn interactions within a compact context window shorter than 4K tokens, enabling deploying search agents in resource-constrained settings.

We summarize our contributions as follows.

- We introduce MemSearcher, an agent framework that leverages the backbone LLM as a memory manager to iteratively retain a compact memory, eliminating the need to append the entire interaction history to LLM contexts.
- We propose multi-context GRPO, which provides end-to-end RL training for MemSearcher trajectories, each of which contains multiple turns under different contexts.
- Comprehensive experiments demonstrate that

MemSearcher achieves superior performance to ReAct-based baselines, while maintaining a nearly constant context length across multi-turn interactions.

## 2 The MemSearcher Framework

In this section, we first revisit the ReAct paradigm and its limitation of ever-growing contexts. Then, we introduce the MemSearcher framework, which maintains a compact context throughout multi-turn interactions via iterative memory integration.

### 2.1 Preliminary: ReAct

ReAct (Yao et al., 2023) has become the most popular paradigm for building LLM-based agents (Jin et al., 2025; Chen et al., 2025b; Mo et al., 2025). The idea of ReAct is simple: a trajectory is a multi-turn conversation, and each turn is an interaction between the agent and the environment. At each turn, the LLM generates a thought, and performs an action, after which the environment provides an observation in response to the performed action.

Specifically, we assume that at the  $i$ -th turn, the agent generates thought  $t_i$ , takes an action  $a_i$ , and receives an observation  $o_i$ . In particular,  $o_0 = q$  represents the observation prior to the first turn, where  $q$  denotes the user’s question. At the  $i$ -th turn, the input to the LLM is as follows:

$$c_i = (q, t_1, a_1, o_1, \dots, t_{i-1}, a_{i-1}, o_{i-1}). \quad (1)$$

Then, the agent generates  $t_i$  and performs the corresponding action  $a_i$ , following policy  $\pi(t_i, a_i | c_i)$ .

Although straightforward, ReAct leads to a continuous increase in the number of tokens in the LLM context, due to its design of appending all interaction history. This increase is linear throughout interactions, placing significant pressure on the inference of LLMs (Hsieh et al., 2024; Wu et al., 2024; Chen et al., 2025a). In addition, in the context of search agents, the observations are passages retrieved by the search engine, which often include substantial noise and information irrelevant to answering the user’s question. This further constrains the performance of ReAct-based search agents, as illustrated in Figure 2 (Left).

Moreover, the linear growth in the number of tokens leads to increased computational and GPU memory overhead. Consequently, more efficient and scalable methods for building search agents need to be explored.

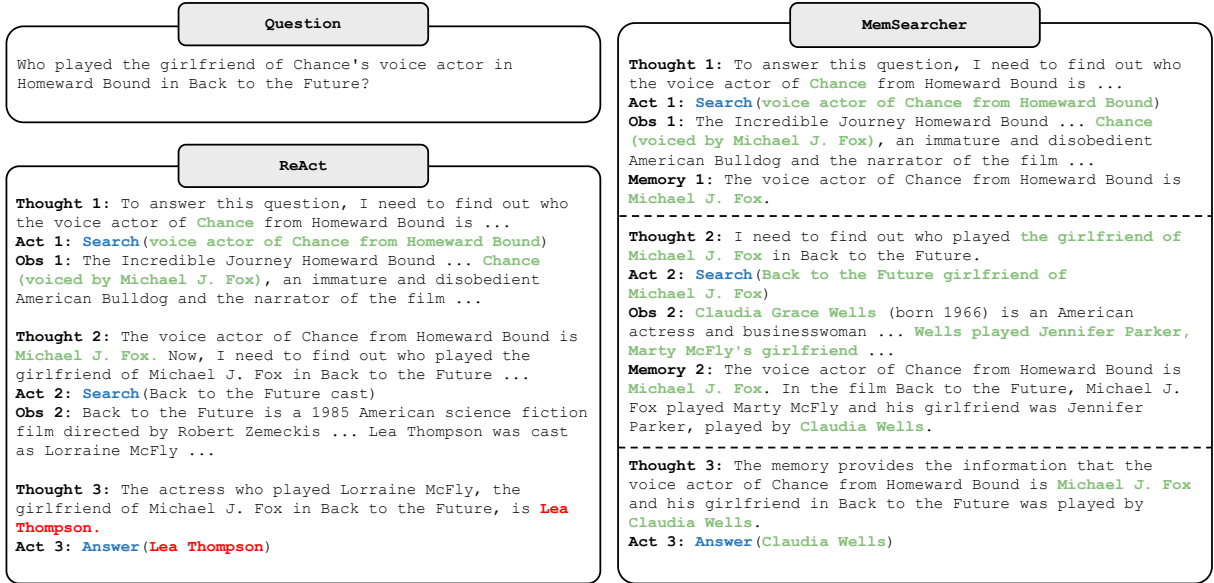


Figure 2: Comparison between ReAct and MemSearcher. ReAct accumulates all intermediate content in the context, leading to an incorrect answer by confusing the character’s girlfriend with the actor’s on-screen relationship. In contrast, MemSearcher maintains an independent context for each turn via a compact memory storing only task-relevant facts, which helps disambiguate entities and relations across turns and yields the correct answer.

## 2.2 Agent with Iterative Memory Integration

The MemSearcher framework is illustrated in Figure 1 (Bottom). At the  $i$ -th turn, the backbone LLM receives only two succinct inputs: the user’s question  $q$  between `<question>` `</question>`, and a compact memory  $m_{i-1}$  expressed in natural language, enclosed within `<memory>` `</memory>`. The memory encapsulates relevant information from previous turns that is considered helpful to answer the question. In particular, the memory  $m_0$  prior to the first turn is empty. Therefore, the input to the LLM at the  $i$ -th turn is formulated as:

$$c_i = (q, m_{i-1}). \quad (2)$$

Upon reading the user’s query and the previous memory, the LLM generates thought  $t_i$  within `<think>` `</think>` and performs an action  $a_i$  between `<tool_call>` `</tool_call>`, following policy  $\pi(t_i, a_i | c_i)$ . As the action is executed, the environment returns the response  $o_i$  within `<tool_response>` `</tool_response>`.

After receiving  $o_i$ , MemSearcher overwrites the previous memory to an updated one for the next turn. The LLM are asked to carefully read  $o_i$  and integrate any new information that helps to answer the question, while preserving all relevant details from the previous memory  $m_{i-1}$ . The resulting memory is denoted as  $m_i$ . This process continues iteratively, until the maximum number of in-

Method	Context	FLOPs / Turn	FLOPs	Memory
ReAct	$O(n)$	$O(n)$	$O(n^2)$	$O(n)$
MemSearcher	$O(1)$	$O(1)$	$O(n)$	$O(1)$

Table 1: Computational overhead comparison, including tokens in context, FLOPs per turn, total FLOPs and GPU memory, scaling with the number of turns  $n$ .

teractions is reached or sufficient information is gathered and the LLM generates a final answer.

Unlike ReAct, which continuously concatenates historical thoughts, actions and observations into the LLM context, MemSearcher compresses only the essential information into a compact memory, as illustrated in Figure 2 (Right).

Although the MemSearcher framework seems simple, it exactly supports the scaling of LLM-based agents. Since the number of tokens in the memory never exceeds a predefined maximum length, MemSearcher maintains the context within a few thousands of tokens. Consequently, the FLOPs per turn keeps  $O(1)$ , and the total compute scales linearly to the number of turns. We illustrate the computational overhead comparison between ReAct and MemSearcher in Table 1.

Specifically, under the setting of search agents with search engines as tools, action  $a_i$  takes one of the following two forms: (1) answer to the user’s question in `\boxed{\}` and terminating the interactions, (2) issuing a search engine call with a

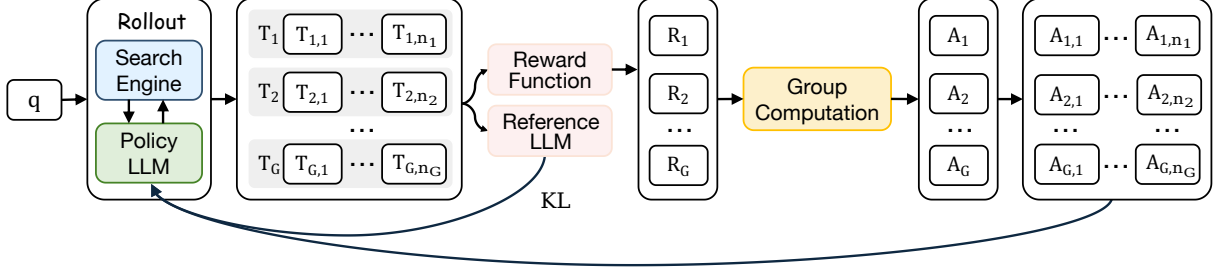


Figure 3: Demonstration of multi-context GRPO. In rollout, it first samples a group of trajectories  $\{T_i\}_{i=1}^G$  for question  $q$ . The  $i$ -th trajectory  $T_i$  consists of multiple turns  $\{T_{i,j}\}_{j=1}^{n_i}$  under different contexts. Then, it computes rewards  $\{R_i\}_{i=1}^G$ , and derives the trajectory-level advantages  $\{A_i\}_{i=1}^G$  from these rewards. It propagates trajectory-level advantages to each turn within them, *i.e.*  $A_{i,j} = A_i$ , and treat each turn as an independent optimization target to update the policy LLM.

query to obtain additional information to answer the query. If the latter is chosen, the returned  $o_i$  is the relevant passages retrieved from the search engine in response to the search query.

### 3 MemSearcher Training

In this section, we introduce the end-to-end RL training method of MemSearcher, including multi-context GRPO and the reward modeling.

#### 3.1 Multi-Context GRPO

We use reinforcement learning (RL) to train MemSearcher agents, since it allows models to evolve without annotated trajectories. Among RL algorithms, we utilize Group Relative Policy Optimization (GRPO) (Shao et al., 2024), as it optimizes the GPU memory usage of Proximal Policy Optimization (PPO) (Schulman et al., 2017).

Vanilla GRPO samples a group of trajectories  $\{T_1, T_2, \dots, T_G\}$  for each query  $q$ , and optimizes policy  $\pi_\theta$  by maximizing the following objective:

$$\mathcal{J}(\theta) = \mathbb{E}_{q \sim D, \{T_i\}_{i=1}^G \sim \pi_{\theta_{old}}(\cdot|q)} \frac{1}{G} \sum_{i=1}^G \left[ \min\left(\frac{\pi_\theta(T_i|q)}{\pi_{\theta_{old}}(T_i|q)} A_i, \text{clip}\left(\frac{\pi_\theta(T_i|q)}{\pi_{\theta_{old}}(T_i|q)}, 1 - \epsilon, 1 + \epsilon\right) A_i\right) - \beta \mathbb{D}_{KL}(\pi_\theta \| \pi_{ref}) \right], \quad (3)$$

where  $A_i$  represents the advantage, calculated with the rewards  $\{R_1, R_2, \dots, R_G\}$  within each group:

$$A_i = \frac{R_i - \text{mean}(\{R_1, R_2, \dots, R_G\})}{\text{std}(\{R_1, R_2, \dots, R_G\})}. \quad (4)$$

For MemSearcher agents, a trajectory consists of multiple turns under different LLM contexts, each of which is an independent optimization target for LLM RL training. Therefore, we introduce multi-context GRPO to enable end-to-end GRPO training for MemSearcher, as illustrated in Figure 3.

Specifically, we assume that trajectory  $T_i$  contains  $n_i$  turns, represented as  $\{T_{i,1}, T_{i,2}, \dots, T_{i,n_i}\}$ . According to Section 2.2, the  $j$ -th turn can be formulated as:

$$T_{i,j} = \begin{cases} (q, m_{i,j-1}, t_{i,j}, a_{i,j}, o_{i,j}, m_{i,j}), & \text{if } j < n_i \\ (q, m_{i,j-1}, t_{i,j}, a_{i,j}), & \text{if } j = n_i \end{cases} \quad (5)$$

where  $m_{i,j-1}$  is the previous memory,  $m_{i,j}$  is the new generated memory,  $t_{i,j}$ ,  $a_{i,j}$  and  $o_{i,j}$  are the reasoning trace, performed action and tool response, respectively.

We compute reward  $R_i$  for each trajectory, and calculate its advantage  $A_i$  within the group by Equation 4. Then, we uniformly propagate this advantage to all turns within the trajectory, and use each turn as an independent target to optimize the policy model. The training objective is:

$$\mathcal{J}(\theta) = \mathbb{E}_{q \sim D, \{T_{i,j}\}_{i=1}^G \sim \pi_{\theta_{old}}(\cdot|c_{i,j})} \frac{1}{\sum_{i=1}^G n_i} \sum_{i=1}^G \sum_{j=1}^{n_i} \left[ \min(r_{i,j}(\theta) A_{i,j}, \text{clip}(r_{i,j}(\theta), 1 - \epsilon, 1 + \epsilon) A_{i,j}) - \beta \text{KL}(\pi_\theta \| \pi_{ref}) \right], \quad (6)$$

where  $c_{i,j} = (q, m_{i,j-1})$  is the input to the model,

$$r_{i,j}(\theta) = \frac{\pi_\theta(T_{i,j}|c_{i,j})}{\pi_{\theta_{old}}(T_{i,j}|c_{i,j})} \text{ and } A_{i,j} = A_i. \quad (7)$$

Notably,  $t_{i,j}$  consists of tokens from both the policy model and the environment. Following previous RL-based search agents (Jin et al., 2025; Chen et al., 2025b), we use loss mask for tokens from the search engine, ensuring the policy gradient objective is computed only over model-generated tokens, thereby stabilizing RL training.

Method	NQ	TriviaQA	PopQA	HotpotQA	2wiki	Musique	Bamboogle	Avg.
<i>Small Size: 3B parameters</i>								
Search-o1 3B (Li et al., 2025)	16.6	31.0	8.2	14.8	22.4	5.2	22.4	17.2
Search-R1 3B-base (Jin et al., 2025)	40.6	58.7	43.5	28.4	27.3	4.9	8.8	30.3
Search-R1 3B-instruct (Jin et al., 2025)	34.1	54.5	37.8	32.4	31.9	10.3	26.4	32.5
AutoRefine 3B-base (Shi et al., 2025)	<u>46.7</u>	<u>62.0</u>	<u>45.0</u>	<u>40.5</u>	<u>39.3</u>	15.7	34.4	<u>40.5</u>
AutoRefine 3B-instruct (Shi et al., 2025)	43.6	59.7	44.7	40.4	38.0	16.9	33.6	39.6
ReSearch 3B (Chen et al., 2025b)	20.4	33.5	17.3	35.6	<u>39.3</u>	<u>17.3</u>	<u>37.6</u>	28.7
ZeroSearch 3B (Sun et al., 2025)	41.4	57.4	44.8	27.4	30.0	9.8	11.1	31.7
O <sup>2</sup> -Searcher 3B (Mei et al., 2025)	44.4	59.7	42.9	38.8	37.4	16.0	34.4	39.1
MemSearcher 3B (Ours)	<b>47.0</b>	<b>63.8</b>	<b>47.9</b>	<b>43.9</b>	<b>43.5</b>	<b>17.9</b>	<b>42.4</b>	<b>43.8</b>
<i>Base Size: 7B parameters</i>								
Search-o1 7B (Li et al., 2025)	19.4	40.6	11.4	17.0	27.0	8.6	30.4	22.1
Search-R1 7B-base (Jin et al., 2025)	<u>48.0</u>	63.8	45.7	43.3	38.2	19.6	<u>43.2</u>	43.1
Search-R1 7B-instruct (Jin et al., 2025)	39.3	61.0	39.7	37.0	41.4	14.6	36.8	38.5
ReSearch 7B (Chen et al., 2025b)	40.9	63.7	44.6	43.5	47.6	<u>22.3</u>	42.4	<u>43.6</u>
ZeroSearch 7B (Sun et al., 2025)	43.6	<u>65.2</u>	<b>48.8</b>	34.6	35.2	18.4	27.8	39.1
R1-Searcher 7B (Song et al., 2025)	40.4	52.2	41.0	<u>44.2</u>	<b>51.3</b>	15.8	36.8	40.2
MemSearcher 7B (Ours)	<b>52.7</b>	<b>68.1</b>	<u>47.8</u>	<b>50.8</b>	<u>48.6</u>	<b>25.8</b>	<b>48.8</b>	<b>48.9</b>
<i>Large Size: &gt; 10B parameters</i>								
Search-o1 14B (Li et al., 2025)	19.8	46.7	23.3	27.4	28.9	11.2	34.4	27.4
Search-R1 14B-base (Jin et al., 2025)	<u>48.6</u>	67.6	48.0	<u>46.8</u>	<u>47.0</u>	24.1	52.8	47.8
Search-R1 14B-instruct (Jin et al., 2025)	42.4	66.0	44.2	43.6	37.9	21.0	48.0	43.3
ReSearch 32B (Chen et al., 2025b)	45.5	<u>69.4</u>	<u>48.2</u>	46.7	44.9	<u>26.4</u>	<u>56.8</u>	<u>48.3</u>
MemSearcher 14B (Ours)	<b>53.7</b>	<b>71.1</b>	<b>48.8</b>	<b>51.8</b>	<b>51.5</b>	<b>27.2</b>	<b>57.6</b>	<b>51.7</b>

Table 2: Performance comparison. The best performance is highlighted in **bold**, while the second-best performance is indicated with an underline.

### 3.2 Reward Modeling

Similar to DeepSeek-R1 (Guo et al., 2025), our reward function considers two parts: format reward and answer reward.

- **Format Reward:** It checks whether the rollout correctly follows the predefined format, including the correctness of usage of tags and the existence of `\boxed{}` in the answer.
- **Answer Reward:** A rule-based reward assesses the correctness of the model’s response. It is calculated by using the F1 score between the final answer inside `\boxed{}` and the ground truth.

The reward function is formulated as:

$$R = \begin{cases} 0, & \text{incorrect format,} \\ 0.1, & \text{correct format \& F1 score}=0, \\ \text{F1 score,} & \text{correct format \& F1 score}>0. \end{cases} \quad (8)$$

## 4 Experiments

To verify the effectiveness of MemSearcher, we conduct experiments on a range of datasets and different scales of LLMs.

### 4.1 Experiment Setups

**Baselines.** We compare MemSearcher against multiple baseline methods, including: (1) prompt-based multi-step inference with search, such as Search-o1 (Li et al., 2025); (2) LLMs with search trained via Reinforcement Learning, such as Search-R1 (Jin et al., 2025), ReSearch (Chen et al., 2025b), AutoRefine (Shi et al., 2025), ZeroSearch (Sun et al., 2025), O<sup>2</sup>-Searcher (Mei et al., 2025), and R1-Searcher (Song et al., 2025).

Among these baselines, ZeroSearch and R1-Searcher interact with the realistic web environment via Google Web Search in their evaluation.

**Benchmarks and Evaluation Metrics.** We compare MemSearcher with the baselines across a range of public benchmarks that encompass search with reasoning challenges, such as Natural Questions (NQ) (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), PopQA (Mallen et al., 2022), Bamboogle (Press et al., 2022), Musique (Trivedi et al., 2022), HotpotQA (Yang et al., 2018), and 2WikiMultiHopQA (Ho et al., 2020). Following Search-R1 (Jin et al., 2025), we use Exact Match (EM) as the evaluation metric.

Model	Method	NQ	TriviaQA	PopQA	HotpotQA	2wiki	Musique	Bamboogle	Average
Qwen2.5-3B	w/o training	16.4	23.8	22.5	11.9	11.0	3.7	11.2	14.4
	-Instruct w/ training	<b>47.0</b>	<b>63.8</b>	<b>47.9</b>	<b>43.9</b>	<b>43.5</b>	<b>17.9</b>	<b>42.4</b>	<b>43.8</b>
Qwen2.5-7B	w/o training	22.1	41.2	23.5	27.4	27.8	11.6	27.2	25.8
	-Instruct w/ training	<b>52.7</b>	<b>68.1</b>	<b>47.8</b>	<b>50.8</b>	<b>48.6</b>	<b>25.8</b>	<b>48.8</b>	<b>48.9</b>
Qwen2.5-14B	w/o training	26.2	53.7	33.2	28.6	24.2	10.1	17.6	27.7
	-Instruct w/ training	<b>53.7</b>	<b>71.1</b>	<b>48.8</b>	<b>51.8</b>	<b>51.5</b>	<b>27.2</b>	<b>57.6</b>	<b>51.7</b>

Table 3: Comparison between models with and without RL training.

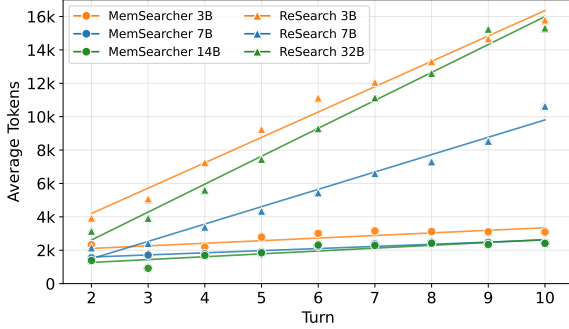


Figure 4: Comparison of the average token numbers in the LLM context between MemSearcher and ReAct-based ReSearch (Chen et al., 2025b).

**Implementation Details.** We conduct our experiments on Qwen2.5-3B/7B/14B-Instruct models (Qwen et al., 2025). We use the 2018 Wikipedia dump (Karpukhin et al., 2020) as the knowledge source and E5 (Wang et al., 2022) as the retriever. The maximum number of tokens in the memory is set to 1,024. For training, we follow Search-R1, using its fully open training datasets, including the training splits of NQ (Kwiatkowski et al., 2019) and HotpotQA (Yang et al., 2018), and the rollout group size of each prompt is set to 5. We conduct the training based on the verl library (Sheng et al., 2025), and the rollout temperature is set to 1.0. For evaluation, we test MemSearcher on seven public datasets, covering both in-domain and out-of-domain scenarios. Such a comprehensive evaluation provides deeper insights into the effectiveness of MemSearcher under varied conditions.

## 4.2 Main Results

### 4.2.1 Performance Analysis

In Table 2, we provide a comprehensive performance comparison between MemSearcher and the baselines across the evaluated benchmarks. Several key observations can be drawn from these results:

(1) **MemSearcher consistently outperforms baselines**, demonstrating the effectiveness of our

method. These performance improvements are consistently observed across both in-distribution datasets such as NQ and HotpotQA, and out-of-distribution datasets, such as TriviaQA, PopQA, 2WikiMultiHopQA, Musique and Bamboogle.

(2) **Remarkably, even with a smaller LLM size, MemSearcher outperforms larger baseline models.** For example, MemSearcher 3B attains an average score of 43.8 across the seven benchmarks, surpassing the baselines with 7B parameters. Moreover, MemSearcher 7B achieves an average score of 48.9, even higher than ReSearch 32B. These results suggest that MemSearcher makes more effective use of model capacity.

(3) **MemSearcher surpasses the baselines that rely on the realistic web search engine.** Specifically, MemSearcher achieves superior performance compared to ZeroSearch and R1-Searcher, both of which depend on Google Web Search to retrieve external information during their evaluation.

### 4.2.2 Efficiency Analysis

In addition to the performance gains, MemSearcher achieves superior efficiency compared to ReAct-based agents, as it eliminates the need to include all interaction history within the LLM context.

To validate this, we record the number of tokens in the LLM contexts of MemSearcher and ReAct-based ReSearch at each turn and calculate their average across the evaluated benchmarks. The results are illustrated in Figure 4.

**Compared to ReSearch, which exhibits an linear increase in token consumption during the interaction process, MemSearcher maintains substantially lower and more stable token counts.** This efficiency gain is primarily attributed to the design of MemSearcher, which iteratively updates a memory as context, preserving only the essential information for answering the given question throughout interactions. Notably, MemSearcher enables multi-turn interactions within a compact context window shorter than 4K tokens. Together

Method	NQ	TriviaQA	PopQA	HotpotQA	2wiki	Musique	Bamboogle	Average
SFT	26.0	34.1	29.3	23.2	26.6	13.9	<b>46.4</b>	28.5
RL	<b>47.0</b>	<b>63.8</b>	<b>47.9</b>	<b>43.9</b>	<b>43.5</b>	<b>17.9</b>	42.4	<b>43.8</b>

Table 4: Comparison between models trained via SFT and RL, based on Qwen2.5-3B-Instruct.

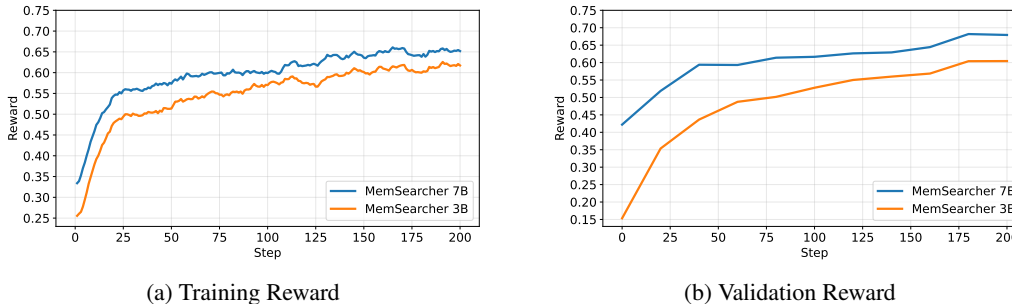


Figure 5: Training and validation reward. The validation is conducted on a part of development set of HotpotQA with 100 randomly selected samples, and conducted every 20 steps. The curves are smoothed for clarity.

with the observation that smaller MemSearcher models even outperform larger baseline models, these results demonstrate the feasibility of deploying search agents in resource-constrained settings.

### 4.3 Further Analysis

#### 4.3.1 Do we need RL Training?

To investigate the impact of RL training on the performance of MemSearcher, we perform a comparative analysis. The baselines are Qwen2.5-3B/7B/14B-Instruct models, both of which are integrated with the MemSearcher framework but do not undergo the RL training.

As shown in Table 3, the models without RL training demonstrate a pronounced performance degradation across all evaluated benchmarks. This observation highlights the necessity of RL training in equipping models with the ability to effectively interact with both the search engine and memory, thereby enhancing their overall functionality and task-solving ability.

#### Why use RL for MemSearcher instead of SFT?

Because SFT relies on token-level supervision and require explicit annotations of intermediate memory states, which are costly. Moreover, many larger LLMs also have not been optimized under the MemSearcher framework, making them suboptimal teachers in distillation-based SFT. In contrast, RL directly rewards behaviors that lead to correct final answers, enabling models to learn what to retain and discard during interactions.

As shown in Table 4, we compare the models trained via SFT and RL. For SFT, we adopt the dis-

tilation mechanism. Specifically, questions from the RL training data are used to prompt Qwen2.5-72B-Instruct, and trajectories with correct final answers are collected to fine-tune Qwen2.5-3B-Instruct. We can observe that the model trained via RL demonstrates superior performance compared to the model with SFT training.

#### 4.3.2 Training and Validation Reward

We present the curves of training and validation reward in Figure 5, which offer an intuitive view of the learning dynamics during training. For validation, we construct a validation dataset by randomly sampling 100 examples from the development set of HotpotQA. We conduct validation every 20 training steps. The observed reward patterns reveals the following two phases of learning:

(1) Early stage (first 25 steps). The reward increases sharply, indicating that the models rapidly acquire the fundamental ability to interact effectively with the search engine and memory.

(2) Later stage (after 25 steps). The reward grows at a more gradual pace. This improvement suggests that the models are refining their strategy, progressively enhancing its capacity to exploit the search engine and manage memory.

The difference between the two stages underscores the transition from basic skill acquisition to the optimization of more advanced behaviors.

#### 4.3.3 Impact of Maximum Memory Length

Choosing an appropriate maximum memory length for MemSearcher agents entails a fundamental trade-off between information capacity and mem-

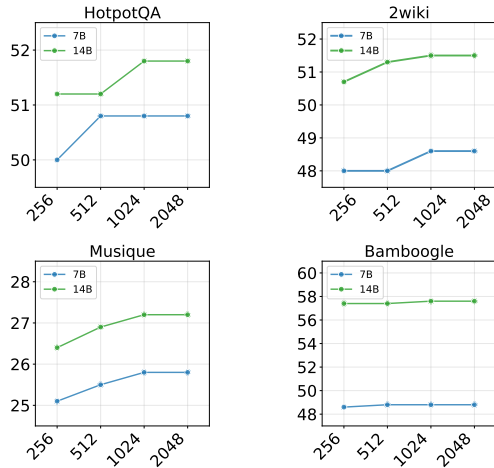


Figure 6: Ablation on maximum memory lengths.

ory efficiency. A larger maximum length enables agents to preserve more information throughout interactions, but it also exacerbates redundancy. In contrast, an overly restrictive memory limit forces the agents to discard essential information, undermining multi-turn search and reasoning performance. In our default configuration, we set the maximum memory length to 1,024 tokens.

To systematically examine the impact of this hyperparameter, we conduct an ablation study with maximum memory lengths ranging from 256 to 2,048 tokens. As shown in Figures 6, the results suggest that intermediate memory sizes provide the most favorable trade-off: they retain sufficient task-relevant information while limiting redundancy in memory integration. On relatively simple datasets such as Bamboogle, MemSearcher already achieves superior and saturated performance with a maximum memory length of 256 tokens, suggesting that MemSearcher is able to compress essential information into a small number of tokens. In contrast, on more complex datasets such as Musique, model performance improves steadily as the maximum memory length increases from 256 to 1,024 tokens.

## 5 Related Work

### 5.1 LLMs with Search Engines

Large Language Models (LLMs) have demonstrated strong and robust capabilities in solving complex tasks (Achiam et al., 2023; Comanici et al., 2025; Men et al., 2025; Yuan et al., 2025a,b; Zeng et al., 2025), yet often lack knowledge in specific domains (Peng et al., 2023; Li et al., 2023; Zhang et al., 2025b; Zheng et al., 2026). Conven-

tional solutions like Retrieval-Augmented Generation (RAG) (Lewis et al., 2020; Yue et al., 2024; Yu et al., 2022) address this by augmenting prompts with retrieved documents. Alternatively, search agents formulate the problem as an interactive process, treating search engines as external tools. ReAct (Yao et al., 2023) integrates search into the reasoning process by interleaving it with Chains-of-Thought (CoT) (Wei et al., 2022) steps. Recent studies (Jin et al., 2025; Chen et al., 2025b) develop agentic reinforcement learning (RL) to enhance search agents. However, current RL-based approaches (Wu et al., 2025; Tao et al., 2025) predominantly adhere to the ReAct paradigm, lacking the exploration of more efficient paradigms.

### 5.2 Context Management

Although context management is pivotal for long-horizon tasks, most LLM agents utilize ReAct (Yao et al., 2023), which leads to prolonged token sequences. To address this, various memory mechanisms have been proposed: (1) RAG-style memory (Jimenez Gutierrez et al., 2024; Zhong et al., 2024) treats memory as an external knowledge source to be retrieved when needed (Zhu et al., 2023). (2) Token-level memory (Jin et al., 2024; Zhou et al., 2025; Orlicki, 2025) uses SFT or RL algorithms (Schulman et al., 2017) to regulate contexts. Strategies range from maintaining latent tokens (Wang et al., 2025), forget-resistant buffers (Yang et al., 2024), or to compressing long-context inputs into summaries (Yu et al., 2025). (3) Structured memory (Zeng et al., 2024; Zhang et al., 2025a; Chhikara et al., 2025) organizes information into structured representations, such as the knowledge graphs utilized in Zep (Rasmussen et al., 2025) and the atomic memory units in A-MEM (Xu et al., 2025). In this paper, we utilize the backbone LLM of the search agent as an intrinsic memory manager, optimizing it via end-to-end multi-context GRPO.

## 6 Conclusion

In this paper, we propose MemSearcher, an agent framework that integrates a memory with the backbone LLM throughout interactions between the agent and the environment. It eliminates the need to append all historical thoughts, actions and observations within the LLM context, as in ReAct. We further introduce multi-context GRPO to optimize MemSearcher-based agents in an end-to-

end fashion. MemSearcher demonstrate superior performance across a range of public benchmarks compared to ReAct-based baselines. Moreover, MemSearcher maintains nearly constant context length shorter than 4K tokens throughout multi-turn interactions with the environment.

## Limitations

Although MemSearcher demonstrates superior performance and efficiency compared to ReAct-based methods, there are several limitations. While MemSearcher adopts a simple and effective memory design, more sophisticated memory mechanisms, such as RAG-like memory or structured memory, need further investigation for RL-based search agents. In addition, Multi-context GRPO serves as an effective extension of the vanilla GRPO algorithm, yet more techniques need to be explored to mitigate potential length bias and to exploit the potential of RL training.

## Acknowledgment

We sincerely thank the reviewers for their insightful comments and valuable suggestions. This work was supported by the National Key R&D Program of China (2024YFC3308000), the Natural Science Foundation of China (No. 62536008, 62306303, 62476265).

## References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Jiawei Chen, Xinyan Guan, Qianhao Yuan, Guozhao Mo, Weixiang Zhou, Yaojie Lu, Hongyu Lin, Ben He, Le Sun, and Xianpei Han. 2025a. Consistentchat: Building skeleton-guided consistent dialogues for large language models from scratch. *arXiv preprint arXiv:2506.03558*.
- Mingyang Chen, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z Pan, Wen Zhang, Huajun Chen, Fan Yang, and 1 others. 2025b. Learning to reason with search for llms via reinforcement learning. *arXiv preprint arXiv:2503.19470*.
- Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.
- Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shitong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.
- Cheng-Ping Hsieh, Simeng Sun, Samuel Krizan, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What’s the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*.
- Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in Neural Information Processing Systems*, 37:59532–59569.
- Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.
- Mingyu Jin, Weidi Luo, Sitao Cheng, Xinyi Wang, Wenyue Hua, Ruixiang Tang, William Yang Wang, and Yongfeng Zhang. 2024. Disentangling memory and reasoning ability in large language models. *arXiv preprint arXiv:2411.13504*.
- Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.
- Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pages 6769–6781.
- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, and 1 others. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.
- Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*.
- Yinheng Li, Shaofei Wang, Han Ding, and Hang Chen. 2023. Large language models in finance: A survey. In *Proceedings of the fourth ACM international conference on AI in finance*, pages 374–382.
- Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.
- Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Hannaneh Hajishirzi, and Daniel Khashabi. 2022. When not to trust language models: Investigating effectiveness and limitations of parametric and non-parametric memories. *arXiv preprint arXiv:2212.10511*, 7.
- Jianbiao Mei, Tao Hu, Daocheng Fu, Licheng Wen, Xueming Yang, Rong Wu, Pinlong Cai, Xinyu Cai, Xing Gao, Yu Yang, and 1 others. 2025. O<sup>2</sup>-searcher: A searching-based agent model for open-domain open-ended question answering. *arXiv preprint arXiv:2505.16582*.
- Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2025. Shortgpt: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 20192–20204.
- Guozhao Mo, Wenliang Zhong, Jiawei Chen, Qianhao Yuan, Xuanang Chen, Yaojie Lu, Hongyu Lin, Ben He, Xianpei Han, and Le Sun. 2025. Livemcpbench: Can agents navigate an ocean of mcp tools? *arXiv preprint arXiv:2508.01780*.
- José I Orlicki. 2025. Beyond words: A latent memory approach to internal reasoning in llms. *arXiv preprint arXiv:2502.21030*.
- Cheng Peng, Xi Yang, Aokun Chen, Kaleb E Smith, Nima PourNejatian, Anthony B Costa, Cheryl Martin, Mona G Flores, Ying Zhang, Tanja Magoc, and 1 others. 2023. A study of generative large language model for medical research and healthcare. *NPJ digital medicine*, 6(1):210.
- Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2022. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*.
- Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jingren Zhou, Junyang Lin, and 24 others. 2025. *Qwen2.5 technical report*. *Preprint, arXiv:2412.15115*.
- Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.
- Yaorui Shi, Sihang Li, Chang Wu, Zhiyuan Liu, Junfeng Fang, Hengxing Cai, An Zhang, and Xiang Wang. 2025. Search and refine during think: Autonomous retrieval-augmented reasoning of llms. *arXiv preprint arXiv:2505.11277*.
- Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. 2025. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*.
- Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. 2025. Zerosearch: Incentivize the search capability of llms without searching. *arXiv preprint arXiv:2505.04588*.
- Richard S Sutton, Andrew G Barto, and 1 others. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.
- Zhengwei Tao, Jialong Wu, Wenbiao Yin, Junkai Zhang, Baixuan Li, Haiyang Shen, Kuan Li, Liwen Zhang, Xinyu Wang, Yong Jiang, and 1 others. 2025. Webshaper: Agentic data synthesizing via information-seeking formalization. *arXiv preprint arXiv:2507.15061*.
- Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

- Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*.
- Yu Wang, Dmitry Krotov, Yuanzhe Hu, Yifan Gao, Wangchunshu Zhou, Julian McAuley, Dan Gutfreund, Rogerio Feris, and Zexue He. 2025. M+: Extending memoryllm with scalable long-term memory. *arXiv preprint arXiv:2502.00592*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2024. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*.
- Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Gang Fu, Yong Jiang, and 1 others. 2025. Web-dancer: Towards autonomous information seeking agency. *arXiv preprint arXiv:2505.22648*.
- Guangzhi Xiong, Qiao Jin, Xiao Wang, Yin Fang, Haolin Liu, Yifan Yang, Fangyuan Chen, Zhixing Song, Dengyu Wang, Minjia Zhang, and 1 others. 2025. Rag-gym: Optimizing reasoning and search agents with process supervision. *arXiv preprint arXiv:2502.13957*.
- Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*.
- Hongkang Yang, Zehao Lin, Wenjin Wang, Hao Wu, Zhiyu Li, Bo Tang, Wenqiang Wei, Jinbo Wang, Zeyun Tang, Shichao Song, and 1 others. 2024. Memory3: Language modeling with explicit memory. *arXiv preprint arXiv:2407.01178*.
- Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.
- Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and 1 others. 2025. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *arXiv preprint arXiv:2507.02259*.
- Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. 2022. Generate rather than retrieve: Large language models are strong context generators. *arXiv preprint arXiv:2209.10063*.
- Qianhao Yuan, Yanjiang Liu, Yaojie Lu, Hongyu Lin, Ben He, Xianpei Han, and Le Sun. 2025a. Saisa: Towards multimodal large language models with both training and inference efficiency. *arXiv preprint arXiv:2502.02458*.
- Qianhao Yuan, Qingyu Zhang, Yanjiang Liu, Jiawei Chen, Yaojie Lu, Hongyu Lin, Jia Zheng, Xianpei Han, and Le Sun. 2025b. Shortv: Efficient multimodal large language models by freezing visual tokens in ineffective layers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 329–339.
- Zhenrui Yue, Honglei Zhuang, Aijun Bai, Kai Hui, Rolf Jagerman, Hansi Zeng, Zhen Qin, Dong Wang, Xuanhui Wang, and Michael Bendersky. 2024. Inference scaling for long-context retrieval augmented generation. *arXiv preprint arXiv:2410.04343*.
- Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.
- Ruihong Zeng, Jinyuan Fang, Siwei Liu, and Zaiqiao Meng. 2024. On the structural memory of llm agents. *arXiv preprint arXiv:2412.15266*.
- Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. 2025a. G-memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*.
- Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, and 1 others. 2025b. Siren’s song in the ai ocean: A survey on hallucination in large language models. *Computational Linguistics*, pages 1–46.
- Hao Zheng, Guozhao Mo, Xinru Yan, Qianhao Yuan, Wenkai Zhang, Xuanang Chen, Yaojie Lu, Hongyu Lin, Xianpei Han, and Le Sun. 2026. Deepresenter: Environment-grounded reflection for agentic presentation generation. *arXiv preprint arXiv:2602.22839*.
- Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731.
- Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. 2025. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, and 1 others. 2023. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*.

## A Appendix

### A.1 LLM Usage

We used OpenAI’s ChatGPT to help polish the language and improve the readability of the manuscript. Specifically, ChatGPT was used for grammar checking and sentence rephrasing. We list our prompt for using OpenAI’s ChatGPT to help polish writing as follows.

```
Prompt for Using OpenAI’s ChatGPT to
Help Polish Writing

Below is a paragraph from an
academic paper. Polish
the writing to meet the
academic style, improve
the spelling, grammar,
clarity, concision and
overall readability.
Furthermore, list all
modification and explain
the reasons to do so in
markdown table. \\
Paragraph: {paragraph}
```

### A.2 Training Hyperparameters

We train MemSearcher agents with full parameter optimization and gradient checkpointing. We show training hyperparameters in Table 5. The 3B/7B experiments are run on 8 H100 GPUs and 14B on 2x8 H100 GPUs.

Parameter	Value
Learning Rate	1e-6
Train Batch Size	256
Number of Training Epochs	1
Number of Rollout	5
Rollout Temperature	1.0
KL Loss Coefficient	0.001
Clip Ratio	0.2

Table 5: Training details of MemSearcher.

### A.3 Details of Evaluated Datasets

We evaluate MemSearcher agents on the following public question answering datasets:

- **Natural Questions (NQ)** (Kwiatkowski et al., 2019), a QA dataset with questions consisting of real anonymized, aggregated queries issued to the Google search engine.
- **TriviaQA** (Joshi et al., 2017), a large scale challenging dataset with relatively complex, compositional questions, requiring more reasoning to find answers.
- **PopQA** (Mallen et al., 2022), a dataset consisting of 14k questions covering factual information in the long tail.
- **Bamboogle** (Press et al., 2022), a manually constructed dataset with multi-hop questions, where all questions are sufficiently difficult to be unanswerable by a single search engine call.
- **Musique** (Trivedi et al., 2022), a QA dataset with 25K questions that by require proper multi-hop reasoning.
- **HotpotQA** (Yang et al., 2018), a dataset with 113k Wikipedia-based question-answer pairs, where the questions requires finding and reasoning over multiple supporting documents to answer, and and not constrained to any existing knowledge bases or knowledge schemas.
- **2WikiMultiHopQA** (Ho et al., 2020), a multi-hop QA dataset that uses both structured and unstructured data. It is challenging for multi-hop models and explicitly ensures that multi-hop reasoning is required.

### A.4 Case Study

To provide a more intuitive understanding of the utility of MemSearcher, we present a case study in Table 6. The text enclosed by <think> and </think>, <tool\_call> and </tool\_call>, as well as <memory> and </memory> is generated by the model. The text enclosed by <tool\_response> and </tool\_response> is retrieved from the search engine. This case demonstrates that the model can effectively maintain a compact memory, retaining only the essential information necessary to solve the question.

---

**Question:** What was the form of the language that the last name Sylvester comes from, used in the era of Rotrude's father, later known as?

**Answer:** Medieval Latin

---

**Turn 1:** <think> To answer this question, I need to find out which language the last name Sylvester comes from. Let's start by searching Wikipedia for this information. </think>

<tool\_call> {"name": "wikipedia\_search", "arguments": {"query": "Last name Sylvester comes from what language"}} </tool\_call>

<tool\_response> Sylvester James Jr. (September 6, 1947 - December 16, 1988), who used the stage name of Sylvester, was an American singer-songwriter. Primarily active in ... .. </tool\_response>

<memory> The last name Sylvester comes from the Latin language. </memory>

---

**Turn 2:** <think> The memory mentions that the last name Sylvester comes from the Latin language. However, it does not provide any information about the form of the Latin language used in the era of Rotrude's father, later known as. I need to find out which era Rotrude's father lived in. </think>

<tool\_call> {"name": "wikipedia\_search", "arguments": {"query": "Who was Rotrude's father, and what was he later known as?"}} </tool\_call>

<tool\_response> Rotrude Rotrude (or sometimes referred to as Hruodrud/Hruodhaid) (775/778 – 6 June 810) was a Frankish princess, the second daughter of Charlemagne from his marriage to Hildegard. Few clear records ... .. </tool\_response>

<memory> The last name Sylvester comes from the Latin language. Rotrude was a Frankish princess, the second daughter of Charlemagne from his marriage to Hildegard. </memory>

---

**Turn 3:** <think> The memory mentions that the last name Sylvester comes from the Latin language. However, it does not provide any information about the form of the Latin language used in the era of Rotrude's father, later known as. I need to find out who Charlemagne was. </think>

<tool\_call> {"name": "wikipedia\_search", "arguments": {"query": "Who was Charlemagne"}} </tool\_call>

<tool\_response> Charlemagne or Charles the Great (2 April 742 – 28 January 814), numbered Charles I, was King of the Franks from 768, King of the Lombards from 774, and Holy Roman Emperor from ... .. </tool\_response>

<memory> The last name Sylvester comes from the Latin language. Rotrude was a Frankish princess, the second daughter of Charlemagne from his marriage to Hildegard. Charlemagne was later known as Holy Roman Emperor. </memory>

---

**Turn 4:** <think> The last name Sylvester comes from the Latin language. Charlemagne was later known as Holy Roman Emperor. The question asks what was the form of the Latin language used in the era of Rotrude's father, later known as. Since Rotrude's father was Charlemagne, and Charlemagne was later known as Holy Roman Emperor, we need to find out what was the form of the Latin language used in his era. I will use the wikipedia\_search function to find this information. </think>

<tool\_call> {"name": "wikipedia\_search", "arguments": {"query": "Form of Latin language used in the era of Charlemagne"}} </tool\_call>

<tool\_response> Medieval Latin was the form of Latin used in Roman Catholic Western Europe during the Middle Ages ... .. </tool\_response>

<memory> The last name Sylvester comes from the Latin language. Rotrude was a Frankish princess, the second daughter of Charlemagne from his marriage to Hildegard. Charlemagne was later known as Holy Roman Emperor. The form of Latin language used in the era of Charlemagne was Medieval Latin. </memory>

---

**Turn 5:** <think> The memory mentions that the form of Latin language used in the era of Charlemagne was Medieval Latin. </think>

The answer to the question is \boxed{Medieval Latin}.

---

Table 6: A case study of MemSearcher.