ASYNCSWARM: ASYNCHRONOUS MULTI-AGENT TRAINING AND INFERENCE FOR INFORMATION SEEKING

Anonymous authors
Paper under double-blind review

ABSTRACT

Applications of large language models (LLMs) have expanded from single-turn question answering to single-agent systems using multi-step reasoning and tools. Multi-agent systems further extend their ability to solve complex tasks. A classic multi-agent system consists of a lead agent that assigns work to parallel sub-agents. We refer to this system as an *LLM swarm*, which is useful for information-seeking tasks that require extensive search to gather and verify evidence from multiple sources. However, blocking inference makes the lead agent wait for the slowest sub-agent. During online reinforcement learning (RL) for LLM swarms, rollouts also wait for sub-agents to finish dozens of tool calls. We propose AsyncSwarm with two parts to address these challenges: *Asynchronous Swarm Inference* lets the lead agent launch new sub-agents as soon as any sub-agent finishes; *Trace-Retrieval Swarm RL* retrieves stored sub-agent responses to replace slow online execution during training. We use the open-source GLM-4.5-Air and GLM-4.7-Flash as the lead and sub-agent models, respectively, and apply RL gradient updates only to the lead model. Compared to blocking baselines, AsyncSwarm achieves $1.6\times$ faster inference with similar performance and $5.8\times$ the RL training throughput. Under the same RL time budget, AsyncSwarm improves over the initial swarm by 10.6 points on WideSearch and 7.6 points on BrowseComp-ZH, compared with gains of only 1.4 and 1.3 points from online RL, respectively. These results show that AsyncSwarm enables more efficient training while achieving substantially better performance.

1 INTRODUCTION

Large language models can act as agents by combining reasoning and tool use to solve complex tasks (Yao et al., 2022). We refer to a setup in which one LLM agent carries out this process as a *single-agent system*. Complex information-seeking tasks often require searching many sources, following clues, and verifying evidence before answering (Wei et al., 2025; Wong et al., 2025). Multi-agent systems (MASs) extend this approach by allowing agents to explore different search paths and check candidate answers in parallel (Wu et al., 2023; Hong et al., 2024). In this work, we focus on *LLM swarms*, where a lead agent delegates work to multiple sub-agents with separate contexts and combines their final responses. This reliance on sub-agent responses creates efficiency challenges in both inference and training.

For inference, blocking workflows make the lead agent wait for an entire batch of sub-agents (Hadfield et al., 2025). With prompted delegation in Claude Code (v2.1.156) on WideSearch, the slowest sub-agent's latency averages $1.9\times$ the mean sub-agent latency. Kimi K2.5's critical-steps formulation also ties each parallel stage to its longest sub-agent branch (Kimi Team, 2026).

For training, multi-agent work focuses mainly on framework design (Wu et al., 2023; Hong et al., 2024), with less attention to reinforcement learning (RL) for swarms. Yet RL has improved LLM reasoning and tool use (Shao et al., 2024; MindDR Team & Li Auto Inc, 2026). It can train the lead agent to decide when to launch sub-agents and how to assign and follow up on tasks. However, a key challenge is the cost of generating rollouts with real sub-agents. In our setting, a rollout can involve

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

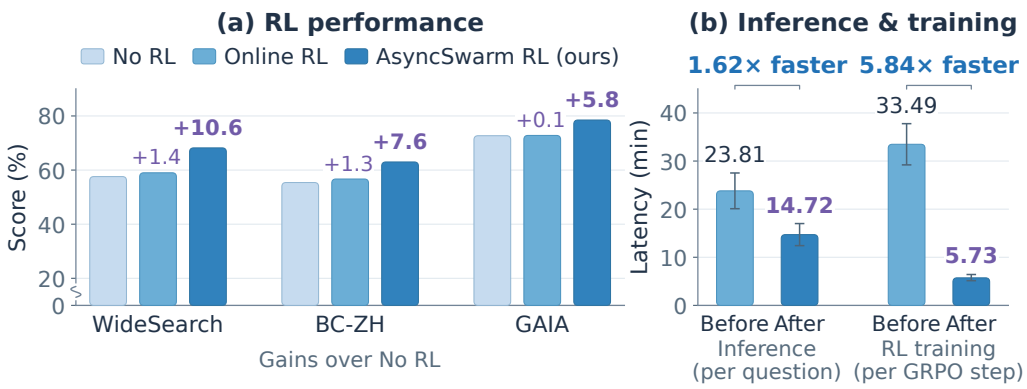


Figure 1: Performance and latency of AsyncSwarm. **(a)** Both RL methods use the same 22-hour budget and hardware. WideSearch uses Avg@4 Item-F1; BrowseComp-ZH (BC-ZH) and text-only GAIA use accuracy. **(b)** Before/After denotes blocking/asynchronous inference and online/trace-retrieval RL. Inference uses the same checkpoint and sub-agent budget; RL timing excludes one-time corpus collection. Error bars show 95% confidence intervals (Appendix E).

dozens of calls, and the lead agent must wait for sub-agent responses needed for its next decision. These delays slow rollouts and limit RL updates within a fixed time budget.

We propose AsyncSwarm for asynchronous swarm training and inference to address blocking delays at inference and costly RL rollouts. It combines *Asynchronous Swarm Inference* with *Trace-Retrieval Swarm RL Training*. To reduce inference latency, the lead agent acts on each sub-agent's final response as it arrives, launching follow-up or verification tasks while other sub-agents continue. To accelerate RL training, the rollout environment retains the asynchronous tool interface and retrieves final responses from a pre-collected trace corpus instead of executing sub-agents online. We train only the lead agent and keep the sub-agent model fixed. Reusing these traces reduces the cost of generating RL rollouts. Even accounting for corpus collection time, training can be faster than with online sub-agents; Appendix F provides the conditions and proof.

Figure 1 summarizes our results with open-source models. Under the same training time budget, Trace-Retrieval Swarm RL delivers much larger gains over the initial swarm than online RL across WideSearch (Wong et al., 2025), BrowseComp-ZH (Zhou et al., 2025), and GAIA (Mialon et al., 2023). Retrieving stored sub-agent responses makes each RL step $5.84\times$ faster than online RL, allowing more updates within that budget. Asynchronous inference is also $1.62\times$ faster than blocking execution with similar performance on WideSearch. Our contributions are:

- **Asynchronous Swarm Inference.** The lead agent uses completed sub-agent responses to start follow-up tasks while other sub-agents continue working.
- **Trace-Retrieval Swarm RL Training.** Our rollout environment keeps the asynchronous interface and retrieves stored sub-agent responses to train the lead agent with RL.
- **Analysis of learned orchestration.** We use controlled comparisons and trajectory analysis to study how RL changes delegation and verification, with the sub-agent and inference loop fixed.

2 The AsyncSwarm Method

We present AsyncSwarm to efficiently train multi-agent swarms with RL and improve their information-seeking performance (Figure 2). The method has two components targeting inference and RL training, respectively: *Asynchronous Swarm Inference* lets the lead agent start follow-up work from completed sub-agent responses while other tasks continue, reducing waiting time at inference. *Trace-Retrieval Swarm RL Training* reuses stored sub-agent responses to avoid repeated online searches and reduce RL rollout cost (Figure 3).

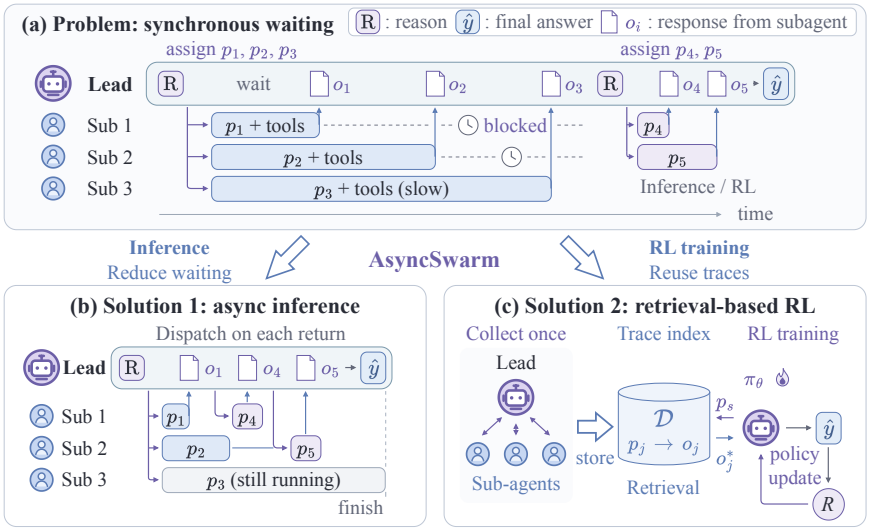


Figure 2: **AsyncSwarm reduces waiting at inference and reuses sub-agent responses during RL. (a)** Blocking execution waits for the slowest sub-agent. **(b)** Asynchronous inference uses early responses for follow-up work and can finish while slower tasks continue. **(c)** Trace-Retrieval Swarm RL collects sub-prompt–final-response pairs once and retrieves responses for new rollouts.

2.1 PRELIMINARIES

Task formulation. We model the lead agent's interaction with the swarm as a finite-horizon *partially observable Markov decision process* (POMDP; Kaelbling et al., 1998), written as

$$\mathcal{M} = \big(\mathcal{X}, \mathcal{A}, P, r, \mathcal{O}, Z, T_{\max}\big).$$

The sub-agents and their tools are part of the environment $\mathcal{M}$. Here, $\mathcal{X}$ is the environment state space, $\mathcal{A}$ the action space, and $\mathcal{O}$ the space of text observations. The models $P(x_{t+1} \mid x_t, a_t)$ and $Z(o_{t+1} \mid x_{t+1}, a_t)$ describe state transitions and observations, respectively; r is the reward function and $T_{\max}$ is the step limit. A task provides a natural-language query q and a reference answer y hidden from the policy. At step t, the lead agent receives a reply o_t from its coordination tools, while the full environment state x_t remains unobserved. The policy uses the history

$$h_t \;=\; \big(q, o_0, a_0, o_1, a_1, \ldots, o_t\big), \tag{1}$$

and samples a text action $a_t \sim \pi_\theta(\cdot \mid h_t)$. A lead-agent rollout $\tau = (o_0, a_0, o_1, \ldots, a_{T-1}, o_T) \sim \mathcal{M}(\cdot; \pi_\theta)$ ends with a final answer or at the step limit $T_{\max}$. We use terminal rewards (Sutton & Barto, 2018): r returns zero before termination and $\tilde{R}(\hat{y}(\tau), y, \tau)$ at the final step (Section 2.3).

Sub-agent execution and traces. The lead agent delegates sub-prompts to multiple sub-agents, each with its own context. For an assigned sub-prompt p_s, sub-agent s chooses search, browsing, or code tools, reads their outputs, and uses the updated local history to choose its next action. For one completed assignment, its local trace is

$$\tau_s^{\text{sub}} = \big(p_s, m_{s,1}, u_{s,1}, \ldots, m_{s,L_s}, u_{s,L_s}, m_s^{\text{final}}\big).$$

Here, $m_{s,j}$ is a sub-agent message that issues tool calls, $u_{s,j}$ contains their outputs, and L_s counts these local tool-use turns. The trace ends with the final message m_s^{final}.

Lead-agent observations and history. The lead agent's coordination tools return information such as agent IDs and task status. Calling `wait` returns the text of a sub-agent's final message m_s^{final}, denoted o_s, together with the source agent s. The sub-agent's intermediate messages and search, browsing, and code outputs remain in its local trace; this full trace is not copied into the lead's context. The lead's history h_t contains the original query q, its own messages and tool calls (including assigned sub-prompts), and replies from its coordination tools, including the final sub-agent responses collected so far. The lead decides its next action based on its history.

2.2 Asynchronous Swarm Inference

The lead agent decides when to assign work, collect a response, and return its final answer. Calling `wait` returns the next completed sub-agent response while other tasks continue running.

Agent actions and state. We extend Kimi K2.5's `create_subagent` and `assign_task` tools (Kimi Team, 2026) with `wait` and `finalize`. The action sets are

$$\begin{aligned}\mathcal{A}_{\text{lead}} &= \{\texttt{create_subagent}, \texttt{assign_task}(s,p), \texttt{wait}, \texttt{finalize}(\hat{y})\},\\ \mathcal{A}_{\text{sub}} &= \{\texttt{search}, \texttt{open}, \texttt{find}, \texttt{python}\}.\end{aligned} \tag{2}$$

At step t, the runtime maintains S_t, the set of created sub-agents, and $R_t \subseteq S_t$, those whose assigned responses have not been collected. An agent with a completed but uncollected response is still in R_t. The lead policy $\pi_\theta(\cdot \mid h_t)$ chooses (a, s, p) from its message history: $a \in \mathcal{A}_{\text{lead}}$ is a tool, s identifies its target when needed, and p carries a task prompt or, for `finalize`, the answer $\hat{y}$. Unused fields are ignored. The budget $N_{\max}$ limits the number of sub-agents created. Assignment requires an available agent in $S_t \setminus R_t$, and `wait` requires at least one pending response. Invalid actions return short feedback before the policy chooses again.

Asynchronous task execution. Algorithm 1 separates task assignment from response collection. The `create_subagent` action creates an idle agent. Calling `assign_task` starts its task and returns control to the lead agent while that task runs. On `wait`, the environment returns the next completed response o_s and its source s, waiting if no result is ready. Collecting this response appends (p_s, o_s) to h and makes s available for reuse. The lead agent can then ask this agent to verify a finding while other searches continue. It can also wait for more evidence or call `finalize` to return $\hat{y}$ and cancel unfinished tasks. Sub-agent s handles prompt p_s in its own context, using tools for web search, page opening, in-page search, and code execution. After execution latency ℓ_s, its final message m_s^{final} is available; the lead agent receives only its text:

Algorithm 1 Asynchronous Swarm Inference

Input: query q, policy π_θ, budget $N_{\max}$
Output: final answer $\hat{y}$

$h \leftarrow (q)$; $S, R \leftarrow \emptyset$; $n \leftarrow 0$
while no final answer **do**
  $(a, s, p) \sim \pi_\theta(\cdot \mid h)$
  if $a =$ `create_subagent` and $n < N_{\max}$ **then**
    create idle s; $S \leftarrow S \cup \{s\}$; $n \leftarrow n + 1$
  else if $a =$ `assign_task` and $s \in S \setminus R$ **then**
    $p_s \leftarrow p$; $R \leftarrow R \cup \{s\}$
    start sub-agent s asynchronously
  else if $a =$ `wait` and $R \neq \emptyset$ **then**
    $(s, o_s) \leftarrow$ `await_next`(R)
    $R \leftarrow R \setminus \{s\}$; $h \leftarrow h \oplus (p_s, o_s)$
  else if $a =$ `finalize` **then**
    $\hat{y} \leftarrow p$; cancel all $s \in R$; $R \leftarrow \emptyset$
return $\hat{y}$

$$o_s \;=\; \text{Text}\big(m_s^{\text{final}}\big)\,, \tag{3}$$

and the query and m collected replies form the response record

$$h_t^{\text{resp}} \;=\; \big(q, (p_{s_1}, o_{s_1}), \ldots, (p_{s_m}, o_{s_m})\big), \tag{4}$$

where $s_1, \ldots, s_m$ index assignments in collection order. The full history h_t in Eq. (1) also includes the lead agent's own messages and replies from its coordination tools; Algorithms 1 and 2 leave these routine updates implicit. The blocking baseline waits for all k sub-agents before the lead agent acts again (Hadfield et al., 2025). Acting on an early response can therefore reduce waiting under the same sub-agent budget, even when total work is unchanged (Figure 2). RL uses the same actions and state updates with retrieved sub-agent responses (Section 2.3).

2.3 Trace-Retrieval Swarm RL Training

Training has two stages (Figure 3): collect task–final-response pairs using real sub-agents, then retrieve those responses during RL. Only the lead agent is updated; the sub-agent model stays fixed.

Trace corpus collection. Using the initial lead policy π_0, we collect K independent rollouts with real sub-agents for each training pair $(q, y) \in \mathcal{Q}_{\text{train}}$,

$$\tau^{(1)}, \ldots, \tau^{(K)} \;\sim\; \mathcal{M}\big(\cdot;\, \pi_0\big), \tag{5}$$

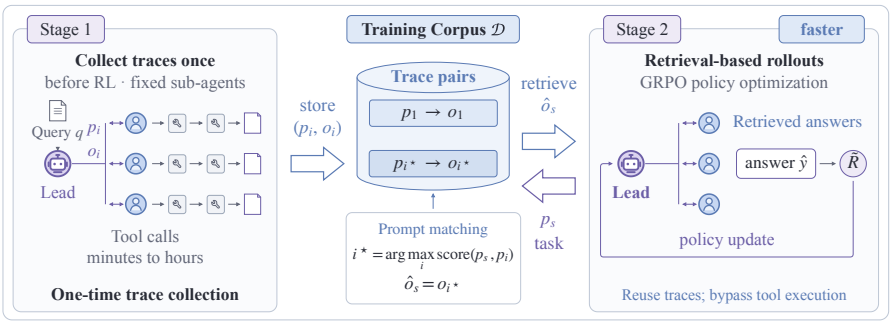


Figure 3: **Two stages of Trace-Retrieval Swarm RL.** *Stage 1*: run real sub-agents and store pairs (p_i, o_i) of subproblems and sub-agent final answers. *Stage 2*: retrieve saved answers for new subproblems p_s and train only the lead agent using reward $\tilde{R}$ from its final answer $\hat{y}$.

with $K = 4$ in our experiments (Appendix A). From each completed local trace τ_s^{sub}, we keep only its sub-prompt p_s and final response o_s for retrieval, giving

$$\mathcal{D} \;=\; \Big\{\big(p_i,\, o_i\big)\Big\}_i, \tag{6}$$

where i indexes completed assignments across collection rollouts. Each p_i is a subproblem and o_i its sub-agent's final answer; $\hat{y}$ denotes the lead's answer to the original query q. The corpus excludes original queries, intermediate steps, and collection rewards. During RL, π_θ generates new lead-agent actions and final answers, whose rewards drive learning. We reuse $\mathcal{D}$ across updates; Appendix F (Corollary 1) gives conditions for savings after at least two epochs, including collection cost.

Retrieval-based rollouts. To train the lead, we reuse stored sub-agent answers to generate RL rollouts without repeating sub-agent searches. The lead agent uses the same tools and state updates as at inference. On `wait`, pending tasks are handled in assignment order. For each task p_s, the retriever finds the stored subproblem with the highest matching score and returns its paired final answer. Matching can use standard scores such as Jaccard similarity, BM25, or cosine similarity between text embeddings. The current policy uses this answer to choose the next lead-agent action. After retrieval, the runtime adds the task and retrieved final answer to the lead's history and marks the task complete. Algorithm 2 summarizes rollout generation; Appendix B details the alternative score functions.

Algorithm 2 Trace-Retrieval Swarm RL

Input: training pairs $\mathcal{Q}_{\text{train}}$, π_0, group size G, $N_{\max}$
Output: trained lead agent policy π_θ

collect $\mathcal{D}$ using π_0 and Algorithm 1, $\pi_\theta \leftarrow \pi_0$
for each GRPO update **do**
  sample a batch $\mathcal{B} \subset \mathcal{Q}_{\text{train}}$
  for $(q, y) \in \mathcal{B}$ and $g = 1, \ldots, G$ **do**
    $h \leftarrow (q)$; $S, R \leftarrow \emptyset$; $n \leftarrow 0$
    while no final answer **do**
      $(a, s, p) \sim \pi_\theta(\cdot \mid h)$
      apply create/assign/finalize updates (Algorithm 1), w/o sub-agent execution
      if $a ==$ `wait` and $R \neq \emptyset$ **then**
        $s \leftarrow$ `next_ready`(R); $R \leftarrow R \setminus \{s\}$
        $i^\star \leftarrow \arg\max_i \text{score}(p_s, p_i)$
        $\hat{o}_s \leftarrow o_{i^\star}$; $h \leftarrow h \oplus (p_s, \hat{o}_s)$
  compute rewards $\tilde{R}$ and update π_θ with GRPO
return π_θ

These steps continue until `finalize`, producing a rollout $\tau \sim \hat{\mathcal{M}}_{\mathcal{D}}(\cdot; \pi_\theta)$. The retriever supplies final responses, while the lead agent decides when to create or reuse agents, assign tasks, check results, wait, and stop. For the same prompt, the retriever returns the same response, so differences between rollouts come from the lead agent's sampled actions. Each response includes its sub-agent identity (Section 2.2), so the lead agent knows its source regardless of arrival order.

Reward design. For a rollout τ with final answer $\hat{y}$ and reference answer y, the reward is

$$\tilde{R}(\hat{y}, y, \tau) \;=\; \begin{cases} 0, & \text{FormatError}(\tau) = 1, \\ R_{\text{answer}}(\hat{y},\, y), & \text{otherwise.} \end{cases} \tag{7}$$

$\text{FormatError}(\tau)$ flags violations of the shared format; R_{answer} is Item-F1 for tables and LLM-judged correctness for short answers. GRPO updates only the lead (Shao et al., 2024).

Table 1: Performance (%) on WideSearch, BrowseComp-ZH (BC-ZH), and text-only GAIA. SA/MA denote single-/multi-agent systems; a/b lists their WideSearch scores. Ctx. is context length; lead/sub denote roles. "/" and "–" mean not reported. Arrows show gains over No RL; AsyncSwarm RL denotes Trace-Retrieval Swarm RL.

Method	Type	Ctx.	WideSearch	BC-ZH	GAIA
General Models					
Gemini-3-Pro (DeepMind, 2025)	SA / MA	/	57.0 / –	51.6	74.8
Claude-Opus-4.5 (Anthropic, 2025)	SA / MA	200k	– / 71.7	62.4	71.5
o3 (OpenAI, 2025c)	SA / MA	200k	52.6 / 57.3	58.1	70.5
GPT-5-high (OpenAI, 2025b)	SA / MA	400k	62.2 / –	63.0	76.4
Kimi-K2-Instruct-1T (Kimi Team, 2025)	SA / MA	128k	54.4 / 61.2	28.8	57.7
Kimi-K2.5 (Kimi Team, 2026)	SA / MA	256k	72.7 / 79.0	62.3	75.9
GLM-4.5-Air (GLM Team, 2025)	SA	128k	61.3	53.1	–
GLM-4.7-Flash (Z.ai, 2026)	SA	200k	45.7	54.7	–
Agent/search methods					
OpenAI Deep Research (OpenAI, 2026)	–	200k	–	42.9	67.4
MindDR-v1.5-32B (MindDR Team, 2026)	MA (3-agent)	/	35.6	46.5	67.1
AgentFold (Qwen3-30B-A3B) (Ye et al., 2025)	SA (folding)	131k	62.1	47.3	67.0
A-MapReduce (Sonnet 4) (Chen et al., 2026)	MA	/	62.2	–	–
A-MapReduce (o3) (Chen et al., 2026)	MA	/	57.3	–	–
Table-as-Search (Gemini 2.5 Flash) (Lan et al., 2026)	MA	64k	52.7	34.9	52.4
Our swarm agent systems					
GLM-4.5-Air (lead) + GLM-4.7-Flash (sub)	MA	128k	57.6	55.4	72.7
+ Online RL	MA	128k	59.0 (↑ 1.4)	56.7 (↑ 1.3)	72.8 (↑ 0.1)
+ AsyncSwarm RL	MA	128k	**68.2** (↑ **10.6**)	**63.0** (↑ **7.6**)	**78.5** (↑ **5.8**)

3 Experiments

3.1 Experimental Setup

Training Data Construction. After filtering and excluding overlaps with held-out benchmarks, we use 378 deep-search and 253 wide-search questions, each with a reference answer. We run Algorithm 1 four times per question with real sub-agents, collecting 2,524 complete swarm trajectories. Their 16,298 pairs of sub-prompts and final sub-agent responses form the retrieval corpus $\mathcal{D}$, collected once and reused across RL updates. Data sources, filtering criteria, and RL training settings are detailed in Appendix A.

Training Framework. We use open-source GLM-4.5-Air (GLM Team et al., 2025) as the lead agent and GLM-4.7-Flash (Z.ai, 2026) as sub-agents. GRPO in SLIME (Zhu et al., 2025) trains only the lead, using groups of $G = 8$ rollouts and the final-answer reward in Section 2. Only tokens generated by the lead agent contribute to the optimization loss. Each rollout can create at most 12 sub-agents and uses a 128k context limit. We select Word Jaccard (Jaccard, 1901) for efficiency and simplicity in RL training, supported by the fixed-policy inference comparison in Section 3.3.

Evaluations and Baselines. We report four-run mean Item-F1 (Avg@4) on *WideSearch* (Wong et al., 2025), which requires table answers from broad web searches, and LLM-judged accuracy on *BrowseComp-ZH* (Zhou et al., 2025) and *GAIA* (Mialon et al., 2023). Table 1 compares our three

Table 2: Scores and latency on WideSearch. Latency is minutes per question or GRPO step; parentheses give speedups over Blocking or Online. Entries are mean ± SD, except training latency: mean ± half the 95% confidence interval width.

	Inference		RL training	
Metric	**Asynchronous**	**Blocking**	**Trace retrieval**	**Online**
Item-F1 (%)	**68.2** ±3.3	66.9 ±2.3	**68.2** ±3.3	59.0 ±3.5
Latency (min)	**14.72** ±1.44 (**1.62**×)	23.81 ±2.33	**5.73** ±0.64 (**5.84**×)	33.49 ±4.28

training variants, evaluated with real sub-agents, with baselines in their reported settings; Appendix E details the metrics, answer judges, and statistical tests.

3.2 Main Performance

Trace retrieval substantially improves multi-agent performance. Table 1 shows substantial gains from trace retrieval. The last three rows use the same lead and sub-agent models; both RL methods train only the lead for 22 hours on the same hardware, excluding corpus collection. Trace retrieval improves the initial swarm by 10.6, 7.6, and 5.8 points on WideSearch, BrowseComp-ZH, and GAIA, respectively, versus 1.4, 1.3, and 0.1 with online RL. On WideSearch and BrowseComp-ZH, these gains are over five times those of online RL, demonstrating the value of cheaper rollouts within a fixed training budget. All three variants use the same asynchronous evaluation with real sub-agents on held-out tasks, showing that training with retrieved responses helps the swarm solve new tasks. After RL training, our open-source swarm also outperforms several strong baselines on WideSearch, including A-MapReduce (o3) by 10.9 points and Table-as-Search (Gemini 2.5 Flash) by 15.5, based on scores reported with each baseline's original evaluation settings.

AsyncSwarm accelerates both inference and RL training. On WideSearch, asynchronous inference reduces mean latency from 23.81 to 14.72 minutes per question, a 1.62× speedup with the same checkpoint and sub-agent budget and similar Item-F1 (Table 2). The lead agent can begin follow-up work as soon as a response arrives while other sub-agents continue their searches. During RL, retrieving stored responses reduces mean GRPO step time from 33.49 to 5.73 minutes, a 5.84× throughput gain on the same hardware. The same swarm thus benefits from faster inference and more RL updates within 22 hours, translating rollout savings into the performance gains in Table 1. Training timings exclude one-time corpus collection. The theoretical cost model in Appendix F includes collection and rollout generation and derives conditions for net savings through reuse across epochs and RL experiments.

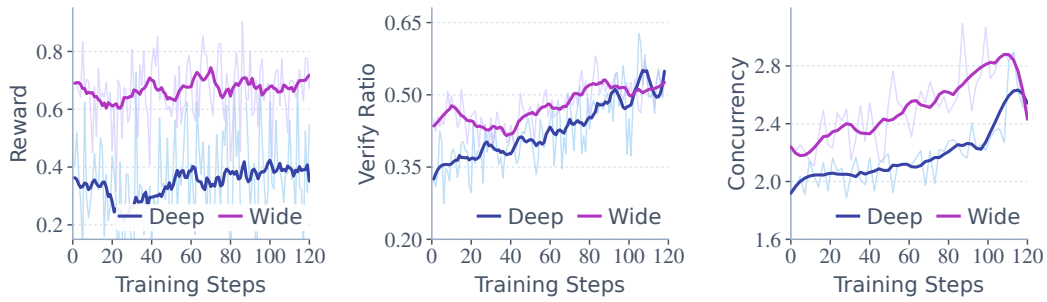
RL changes coordination during training. During RL on the deep and wide data, the lead agent asks for more checks and assigns more tasks in parallel as rewards improve (Figure 4). RL uses only the final-answer reward, without a separate verification reward or updates to the sub-agent model.

Table 3: Ablation of deep and wide training data on three information-seeking benchmarks.

Variant	WideSearch	BC-ZH	GAIA
AsyncSwarm	**68.2**	63.0	**78.5**
Deep only	60.0 ↓ 8.2	**63.1** ↑ 0.1	78.2 ↓ 0.3
Wide only	64.8 ↓ 3.4	54.3 ↓ 8.7	70.8 ↓ 7.7
No RL	57.6 ↓ 10.6	55.4 ↓ 7.6	72.7 ↓ 5.8

Table 4: Ablation of retrievers at inference on wide and deep training data (mean ± SD, %).

Retriever	Wide data	Deep data
Word Jaccard	46.5 ± 3.3	**27.9** ± 3.2
Subword	49.8 ± 3.6	27.5 ± 3.5
BM25	**49.9** ± 3.5	25.6 ± 3.5
Dense	45.7 ± 3.6	27.5 ± 3.5



(a) Reward (b) Verify Ratio (c) Concurrency

Figure 4: Training dynamics of the lead agent on the deep and wide data. Reward is the group mean; Verify Ratio measures verification-style tool use; Concurrency is the mean number of parallel `assign_task` calls per delegation round. Light lines show batch values; dark lines show smoothed trends. Raw rewards below 0.15 are outside the displayed range.

3.3 Ablation Study

Ablation of training data. Table 3 compares the initial swarm (No RL) with AsyncSwarm (mixed), deep-only, and wide-only training on three held-out benchmarks, using real sub-agents. The trained variants share the initial lead model and RL budget; arrows show changes from mixed training. Mixed training improves over No RL by 10.6, 7.6, and 5.8 points on WideSearch, BC-ZH, and GAIA. Deep-only scores higher than Wide-only on BC-ZH and GAIA, while Wide-only does better on WideSearch. Combining both data types improves WideSearch by 8.2 points over Deep-only and 3.4 points over Wide-only, with the highest GAIA score and just 0.1 points below Deep-only on BC-ZH.

Ablation of retrievers. Given the computational budget, we limit retriever ablation to inference and do not repeat RL training with each retriever. Table 4 keeps the lead checkpoint, corpus, and question set fixed. We evaluate three runs on corpus-covered wide and deep training data with retrieved responses, using Item-F1 and binary correctness, respectively (Appendix C). BM25 has the highest mean on wide data and Word Jaccard on deep data, so no retriever leads on both. The 95% confidence intervals overlap on both datasets (Appendix E). These results motivate our choice of Word Jaccard for RL training: it combines competitive inference performance on both data types with computational efficiency and a simple implementation.

3.4 Discussions

Table 5: Real versus retrieved sub-agent responses on wide training questions, using the same 16,298-pair corpus $\mathcal{D}$ as RL. Max@k is the mean per-question maximum Item-F1 over k rollouts (%); Δ is the gain from one to four rollouts (Appendix G).

Responses	max@1	max@2	max@4	Gap Δ
Real sub-agents	57.2	67.5	73.2	16.0
Retrieved responses	56.3	66.4	72.0	15.7

Trace retrieval supports effective rollouts on wide training data. Table 5 compares rollouts using real sub-agents or retrieved responses on wide training questions. Retrieval reuses the same 16,298-pair corpus $\mathcal{D}$ used for RL training. We report max@1, max@2, and max@4 Item-F1, averaging each question's highest score over k rollouts (Appendix G). Retrieved rollouts remain within 0.9–1.2 points of real execution, with similar gains from one to four samples (15.7 versus 16.0 points). These results support using pre-collected responses for offline rollouts on wide training questions covered by the corpus. The diagnostic uses four sampled answers; GRPO uses groups of eight. Table 1 evaluates generalization separately with real sub-agents on held-out benchmarks.

RL changes lead-agent behavior and generalizes to new tasks. On held-out WideSearch and BrowseComp-ZH evaluated with real sub-agents, RL increases the share of rollouts containing at least one verification sub-prompt (Appendix D, Table 7). This verification rate rises from 87.2% to 100% on WideSearch and from 60.2% to 99% on BrowseComp-ZH. WideSearch allocates a larger share of all delegated tasks to verification, while BrowseComp-ZH verifies more rollouts with a smaller task share. Tasks per question rise from 7.37 to 8.50 on WideSearch and from 8.95 to 9.45 on BrowseComp-ZH, while sub-agents per question fall from 6.28 to 2.60 and from 5.58 to 2.59. More assignments with fewer sub-agents indicate greater reuse of existing contexts. Task instructions also become longer, especially on WideSearch, consistent with its need for broader evidence collection. These coordination changes accompany higher held-out scores with real sub-agents.

4 Related Work

Multi-Agent Frameworks for LLM Agents. Multi-agent frameworks extend single-agent tool use (Yao et al., 2022) to teams. CAMEL (Li et al., 2023), AutoGen (Wu et al., 2023), and MetaGPT (Hong et al., 2024) organize agent collaboration. A-MapReduce (Chen et al., 2026) and Table-as-Search (Lan et al., 2026) support broad searches through map-reduce and table completion. Anthropic's report (Hadfield et al., 2025) describes batch execution, where progress depends on the slowest sub-agent (Dean & Barroso, 2013). Claude Code (Anthropic, 2026) supports asynchronous completion notifications; AutoGen Core (Microsoft, 2026) uses asynchronous messaging; and AgentScope (Gao et al., 2024) avoids blocking with placeholder messages. DynTaskMAS (Yu et al., 2025a) organizes dependent tasks with dynamic graphs, while Gradientsys (Song et al., 2025) combines synchronous and asynchronous dispatch with a ReAct planner.

Reinforcement Learning for Information Seeking Tasks. RL is a key paradigm for LLM post-training. Common RL algorithms include GRPO (Shao et al., 2024), which compares rewards within groups of answers, and DAPO (Yu et al., 2025b), which improves training stability and efficiency through dynamic sampling and decoupled clipping. MindDR (MindDR Team & Li Auto Inc, 2026) trains agents for multi-step web search. AgentFold (Ye et al., 2025) manages context during long searches, and Kimi K2.5 (Kimi Team, 2026) trains a lead agent with fixed sub-agents. Asynchronous RLHF (Noukhovitch et al., 2024), AReaL (Fu et al., 2025), and ROLL Flash (Lu et al., 2025) reduce waiting in RL pipelines. Offline RL (Levine et al., 2020) reuses collected experience, while retrieval-augmented generation (Lewis et al., 2020) uses retrieved text to help generate answers.

5 Conclusion

We presented AsyncSwarm for asynchronous multi-agent information seeking. Using completed sub-agent responses reduces inference waiting, while retrieving stored responses makes RL rollouts faster and better. Inference is $1.62\times$ faster with similar Item-F1, and RL throughput increases $5.84\times$ after corpus collection. Within the same 22-hour budget, the trained lead agent achieves higher search scores than blocking online RL. Behavioural analysis shows more consistent checks and greater reuse of sub-agent contexts, with the sub-agent model fixed. These results support collecting sub-agent responses once and reusing them for repeated lead-agent updates, while preserving the original asynchronous interaction with real sub-agents at inference.

6 Limitations

Our evaluation covers information seeking with parallel search and verification. Benefits may be smaller for tasks dominated by internal reasoning, long interaction histories, or writing within one context. Retrieval depends on corpus coverage and may return unsuitable responses for unseen sub-prompts. Code changes in SWE-bench Verified, for example, depend on the current codebase and may require fresh sub-agent execution.

References

Anthropic. A statistical approach to model evaluations. `https://www.anthropic.com/research/statistical-approach-to-model-evals`, 2024. Accessed: 2026-06-02.

Anthropic. Claude Opus 4.5 system card. https://www.anthropic.com/system-cards, 2025. System card, November 2025. Accessed: 2026-06-11.

Anthropic. Create custom subagents — Claude Code docs. https://code.claude.com/docs/en/sub-agents, 2026. Accessed: 2026-09-08; section: Run subagents in foreground or background.

Mingju Chen, Guibin Zhang, Heng Chang, Yuchen Guo, and Shiji Zhou. A-MapReduce: Executing wide search via agentic mapreduce. *arXiv preprint arXiv:2602.01331*, 2026. URL https://arxiv.org/abs/2602.01331.

Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 1383–1392, 2018. doi: 10.18653/v1/P18-1128. URL https://aclanthology.org/P18-1128/.

Morten W. Fagerland, Stian Lydersen, and Petter Laake. The McNemar test for binary matched-pairs data: Mid-p and asymptotic are better than exact conditional. *BMC Medical Research Methodology*, 13(91), 2013.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. AReaL: A large-scale asynchronous reinforcement learning system for language reasoning. *arXiv preprint arXiv:2505.24298*, 2025.

Dawei Gao, Zitao Li, Xuchen Pan, Weirui Kuang, Zhijian Ma, Bingchen Qian, Fei Wei, Wenhao Zhang, Yuexiang Xie, Daoyuan Chen, Liuyi Yao, Hongyi Peng, Zeyu Zhang, Lin Zhu, Chen Cheng, Hongzhu Shi, Yaliang Li, Bolin Ding, and Jingren Zhou. AgentScope: A flexible yet robust multi-agent platform. *arXiv preprint arXiv:2402.14034*, 2024. URL https://arxiv.org/abs/2402.14034.

GLM-5 Team. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026. URL https://arxiv.org/abs/2602.15763.

GLM Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. GLM-4.5: Agentic, reasoning, and coding (ARC) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

Google DeepMind. Gemini 3 Pro. https://deepmind.google/models/model-cards/gemini-3-pro/, 2025. Model card, November 2025. Accessed: 2026-07-26.

Jeremy Hadfield, Barry Zhang, Kenneth Lien, Florian Scholz, Jeremy Fox, and Daniel Ford. How we built our multi-agent research system. Anthropic Engineering, https://www.anthropic.com/engineering/multi-agent-research-system, June 2025.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024.

Paul Jaccard. Étude comparative de la distribution florale dans une portion des Alpes et du Jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

Kimi Team. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Kimi Team. Kimi K2.5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026. URL https://arxiv.org/abs/2602.02276.

Tian Lan, Felix Henry, Bin Zhu, Qianghuai Jia, Junyang Ren, Qihang Pu, Haijun Li, Longyue Wang, Zhao Xu, and Weihua Luo. Table-as-Search: Formulate long-horizon agentic information seeking as table completion. *arXiv preprint arXiv:2602.06724*, 2026. URL https://arxiv.org/abs/2602.06724.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

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. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. *arXiv preprint arXiv:2303.17760*, 2023.

Han Lu, Zichen Liu, Shaopan Xiong, Yancheng He, Wei Gao, Yanan Wu, Weixun Wang, Jiashun Liu, Yang Li, Haizhou Zhao, et al. Part II: ROLL Flash — accelerating RLVR and agentic training with asynchrony. *arXiv preprint arXiv:2510.11345*, 2025.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*, 2023.

Microsoft. Core — AutoGen documentation. https://microsoft.github.io/autogen/stable/user-guide/core-user-guide/index.html, 2026. Accessed: 2026-09-08.

MindDR Team and Li Auto Inc. Mind DeepResearch technical report. *arXiv preprint arXiv:2604.14518*, 2026. URL https://arxiv.org/abs/2604.14518.

Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron Courville. Asynchronous RLHF: Faster and more efficient off-policy RL for language models. *arXiv preprint arXiv:2410.18252*, 2024.

OpenAI. BrowseComp evaluation in OpenAI Simple Evals. https://github.com/openai/simple-evals, 2025a. Official evaluation repository. Accessed: 2026-06-23.

OpenAI. GPT-5 system card. https://openai.com/index/gpt-5-system-card/, 2025b. System card. Accessed: 2026-06-24.

OpenAI. OpenAI o3 model documentation. https://platform.openai.com/docs/models/o3, 2025c. OpenAI API documentation. Accessed: 2026-06-24.

OpenAI. Deep Research. https://developers.openai.com/api/docs/guides/deep-research, 2026. OpenAI documentation. Accessed: 2026-09-18.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 2009. doi: 10.1561/1500000019.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Xinyuan Song, Zeyu Wang, Siyi Wu, Tianyu Shi, and Lynn Ai. Gradientsys: A multi-agent LLM scheduler with ReAct orchestration. *arXiv preprint arXiv:2507.06520*, 2025. URL https://arxiv.org/abs/2507.06520.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. BrowseComp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

Ryan Wong, Jiawei Wang, Junjie Zhao, Li Chen, Yan Gao, Long Zhang, Xuan Zhou, Zuo Wang, Kai Xiang, Ge Zhang, Wenhao Huang, Yang Wang, and Ke Wang. WideSearch: Benchmarking agentic broad info-seeking. *arXiv preprint arXiv:2508.07999*, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Siheng Chen, Jingren Zhou, and Yong Jiang. AgentFold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*, 2025. URL https://arxiv.org/abs/2510.24699.

Alexander Yeh. More accurate tests for the statistical significance of result differences. In *COLING 2000 Volume 2: The 18th International Conference on Computational Linguistics*, 2000. URL https://aclanthology.org/C00-2137/.

Junwei Yu, Yepeng Ding, and Hiroyuki Sato. DynTaskMAS: A dynamic task graph-driven framework for asynchronous and parallel LLM-based multi-agent systems. *arXiv preprint arXiv:2503.07675*, 2025a. URL https://arxiv.org/abs/2503.07675.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025b. URL https://arxiv.org/abs/2503.14476.

Z.ai. GLM-4.7-Flash. https://huggingface.co/zai-org/GLM-4.7-Flash, 2026. Hugging Face model card.

Peilin Zhou, Bruce Leon, Xiang Ying, et al. BrowseComp-ZH: Benchmarking web browsing ability of large language models in chinese. *arXiv preprint arXiv:2504.19314*, 2025.

Yutao Zhu, Xingshuo Zhang, Maosen Zhang, Jiajie Jin, Liancheng Zhang, Xiaoshuai Song, Kangzhi Zhao, Wencong Zeng, Ruiming Tang, Han Li, Ji-Rong Wen, and Zhicheng Dou. GISA: A benchmark for general information-seeking assistant. *arXiv preprint arXiv:2602.08543*, 2026.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

A TRAINING DATA AND RL SETTINGS

Table 6: RL settings for AsyncSwarm with SLIME (Zhu et al., 2025). Only the lead agent is trained. Parallelism: TP (tensor), PP (pipeline), CP (context), EP (expert), and ETP (expert-tensor).

Category	Setting	Value
	Model and data	
Lead model	Open-source model	GLM-4.5-Air
Sub-agent model	Open-source model	GLM-4.7-Flash
Model size	Lead / sub	100B / 30B
RL data	GISA / WebQA questions	253 / 378
Agent limit	Maximum sub-agents	12
	Rollout generation	
Sampling group	Samples per prompt	8
Batch size	Global / rollout	128 / 16
Rollout limit	Maximum rollouts	3,000,000
Token limits	Context / prompt / response	128k / 112k / 16k
Sampling	Temperature / top-p	1.0 / 1.0
Weight updates	Actor update interval	4 steps
	RL objective	
RL method	Algorithm	GRPO
Adv. normalization	Standard deviation	enabled
KL term	Coefficient / type	0.00 / low-variance
Entropy term	Coefficient	0.00
Clipping	Lower / upper	0.3 / 1.8
Reward	Type	final answer + format constraint
	Optimizer	
Optimizer	Type	Adam
Learning rate	Schedule	2×10^{-6} / constant
Adam moments	β_1 / β_2	0.9 / 0.98
Weight decay	Coefficient	0.1
Dropout	Attention / hidden	0.0 / 0.0
	Distributed training and serving (NVIDIA H-series GPUs)	
Training GPUs	Nodes / GPUs per node	4 / 8
Training GPUs	Total GPUs	32
Rollout GPUs	Total GPUs / GPUs per engine	64 / 4
SGLang serving	Memory fraction	0.75
Model parallelism	TP / PP / CP / EP / ETP	8 / 1 / 4 / 16 / 1
Dynamic batching	Max tokens per GPU	25,000
	Memory and checkpointing	
Activation recompute	Granularity / method / layers	full / uniform / 1
Checkpointing	Save interval	8 steps
State reload	Optimizer / RNG	disabled / disabled

Data construction and filtering. For deep search, we use short-answer WebQA questions constructed following GLM-5's data-synthesis procedure (GLM-5 Team, 2026). It builds a knowledge graph from web pages, samples seed entities, and expands their multi-hop neighborhoods into subgraphs for question generation. We keep questions that GLM-4.7 answers correctly in 2–6 of 8 attempts, following pass-rate filtering (Yu et al., 2025b). This leaves 1,766 candidates, from which we randomly sample 378 questions to balance the sources. For wide search, we use 253 of GISA's 373 questions (Zhu et al., 2026), which ask for tables of facts. Before trace collection, we exclude all QA items and associated traces from the held-out WideSearch, BrowseComp-ZH, and GAIA text-only evaluation sets. No train–test overlap is found, including with 5-gram keyword matching.

Trace collection and reuse. For each of the 631 training questions, we independently run Algorithm 1 four times, using GLM-4.5-Air as the lead agent and GLM-4.7-Flash as the fixed sub-agent model. This gives $4 \times 631 = 2{,}524$ complete swarm trajectories. From each completed sub-agent assignment, we retain its subproblem p_i and final answer o_i, obtaining 12,066 WebQA (deep) and 4,232 GISA (wide) pairs. Together, these form the 16,298-pair retrieval corpus $\mathcal{D}$, which we collect once and reuse for trace-retrieval RL runs and ablations.

Training implementation. All RL comparisons use the same NVIDIA H-series GPU allocation (Table 6). Our Word Jaccard retriever (Appendix B) returns a stored sub-agent answer in less than 1.67×10^{-5} minutes. Only lead-agent-generated tokens contribute to the optimization loss; tool outputs and sub-agent responses are excluded.

B Retrieval Procedure and Scoring Functions

During RL, the retriever matches each new subproblem to a stored subproblem and returns its paired answer. The lead agent uses this answer to continue its rollout. Our main RL runs use Word Jaccard, a simple measure of word overlap. Appendix C describes the fixed-checkpoint evaluation of these scores in Table 4.

Retrieval procedure. All retrievers compare the new subproblem p_s with stored subproblems p_i in $\mathcal{D} = \{(p_i, o_i)\}_i$; the corresponding sub-agent answers o_i are not used to compute scores. Algorithm 2 selects the pair with the highest score and returns its answer:

$$i^\star \;=\; \arg\max_i \; \mathrm{score}\big(p_s,\, p_i\big), \qquad \hat{o}_s \;=\; o_{i^\star}. \tag{8}$$

Here $i^\star$ identifies the selected stored pair, while s identifies the sub-agent assigned the current task. Larger scores indicate closer matches, and ties are resolved deterministically.

Word and subword Jaccard. Let $T_\phi(p)$ be the set of distinct tokens in prompt p under representation ϕ. *Word Jaccard* represents prompts as sets of word tokens, while *Subword Jaccard* uses token IDs from a fixed subword tokenizer. Both use Jaccard similarity (Jaccard, 1901):

$$J_\phi(p_s, p_i) = \frac{|T_\phi(p_s) \cap T_\phi(p_i)|}{|T_\phi(p_s) \cup T_\phi(p_i)|}, \qquad \mathrm{score}(p_s, p_i) = J_\phi(p_s, p_i). \tag{9}$$

The corresponding Jaccard distance is $d_\phi(p_s, p_i) = 1 - J_\phi(p_s, p_i)$. Algorithm 2 maximizes similarity, which is equivalent to minimizing this distance. These representations change token granularity while keeping the scoring rule fixed.

Word BM25. Word BM25 (Robertson & Zaragoza, 2009) scores matches using word counts and prompt lengths. Let $f(w, p_i)$ be the number of occurrences of word w in stored prompt p_i, L_i its word count, and $\bar{L}$ the mean word count across stored prompts. The score is

$$\mathrm{score}(p_s, p_i) = \sum_{w \in T_{\mathrm{word}}(p_s)} \mathrm{IDF}(w) \frac{f(w, p_i)(k_1 + 1)}{f(w, p_i) + k_1(1 - b + bL_i/\bar{L})}, \tag{10}$$

where k_1 controls term-frequency saturation and b controls length normalization. The inverse document frequency gives more weight to words that occur in fewer stored prompts:

$$\mathrm{IDF}(w) = \log\left(1 + \frac{|\mathcal{D}| - n_w + 0.5}{n_w + 0.5}\right). \tag{11}$$

Here n_w counts the stored prompts containing w.

Dense cosine. Dense cosine encodes each prompt with a fixed text encoder e and scores the resulting vectors by cosine similarity:

$$\text{score}(p_s, p_i) = \frac{e(p_s)^\top e(p_i)}{\|e(p_s)\|_2 \, \|e(p_i)\|_2}. \tag{12}$$

Stored prompts are encoded once; new prompts are encoded during rollouts. The highest-scoring stored prompt supplies the answer returned to the lead agent.

C Retriever Ablation Protocol

Given the computational budget, Table 4 compares retrievers only at inference, without repeating RL training with each retriever. Appendix B defines the four score functions.

Fixed-checkpoint evaluation. We fix the lead checkpoint, stored subproblem–answer pairs, and question set, replacing only the matching function. For each retriever, we evaluate the same corpus-covered GISA (wide) and WebQA (deep) questions three times, sampling one rollout per question in each run. Each `wait` retrieves the stored answer whose subproblem best matches the pending one, replacing a real sub-agent response. We score the lead agent's final answer with table Item-F1 on GISA and binary correctness on WebQA; questions without a final answer receive zero. Table 4 reports the mean and sample standard deviation over the three runs, following the uncertainty calculation in Appendix E. This comparison measures the effect of retriever choice on a fixed policy; its effect on policy learning during RL remains untested.

D Behavioural Generalization on Test Sets

Verification at the rollout and task levels. Table 7 reports how the learned behavior transfers to held-out WideSearch and BrowseComp-ZH tasks, evaluated with real sub-agents. We identify a verification task by matching verification-related keywords in the sub-prompt assigned by the lead agent. The rollout-level verification rate is the fraction of evaluated rollouts containing at least one such task. The verification task share is the total number of verification tasks divided by all delegated tasks, pooling tasks across rollouts. These ratios measure checking behavior, not evaluation completion. The rollout-level rate rises from 87.2% to 100% on WideSearch and from 60.2% to 99% on BrowseComp-ZH, with both gains supported by paired tests (Dror et al., 2018; Yeh, 2000) (Appendix E). WideSearch also devotes a larger share of tasks to verification, while BrowseComp-ZH verifies more rollouts with a smaller task share. These changes arise under final-answer feedback without an explicit verification reward.

Table 7: How RL changes the lead agent's work on 200 WideSearch and 289 BrowseComp-ZH questions. Task score is either Item-F1 or LLM-judged accuracy. Verification tasks contain matching keywords in their sub-prompts. Δ = After − Before; changes in percentage rows are in percentage points. Paired tests: $^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$; no star means $p \geq 0.05$.

	WideSearch			BrowseComp-ZH		
Metric	Before	After	Δ	Before	After	Δ
Task score (%)	57.6	68.2	$+10.6^{***}$	55.4	63.0	$+7.6^{***}$
Rollouts with verification (%)	87.2	100.0	$+12.8^{***}$	60.2	99.0	$+38.8^{***}$
Verification task share (%)	30.5	38.6	$+8.1^{***}$	19.7	16.8	-2.9^{***}
Tasks per question	7.37	8.50	$+1.13^{***}$	8.95	9.45	$+0.50^{*}$
Prompt tokens per task	254.8	321.4	$+66.6^{***}$	200.2	214.0	$+13.8^{**}$
Sub-agents per question	6.28	2.60	-3.68^{***}	5.58	2.59	-2.99^{***}

Delegation and transfer to real execution. RL uses pre-collected WebQA and GISA traces to simulate sub-agent responses, whereas evaluation runs real sub-agents and tools on held-out WideSearch and BrowseComp-ZH questions. After corpus collection, this yields $5.84\times$ faster training than online RL, while the resulting behaviours transfer to these information-seeking tasks. Table 7 shows score gains of 10.6 and 7.6 points, respectively, along with longer task instructions. More assignments across fewer sub-agents also indicate greater reuse of existing contexts. WideSearch shows larger increases in instruction length and assignments per question, consistent with its need for broader evidence collection than BrowseComp-ZH.

E Evaluation Protocols and Statistical Methods

Benchmark metrics and subsets. For WideSearch (Wong et al., 2025), each cell of the answer table is an item. Item-F1 is the F1 score between the multisets of predicted and gold items, and Avg@4 averages this score over four independent runs. BrowseComp-ZH (Zhou et al., 2025) and GAIA (Mialon et al., 2023) report LLM-judged accuracy. Because our system uses text and web search, we evaluate GAIA on the public 103-task text-only validation subset. All GAIA results for our systems and the compared baselines use this subset.

Answer judging and calibration. For BrowseComp-ZH and GAIA text-only, we use the official BrowseComp answer-equivalence template and evaluation repository (Wei et al., 2025; OpenAI, 2025a) with `o3-mini`, a fixed judge prompt, and deterministic judge decoding. BrowseComp-ZH accuracy uses the automatic judgments without manual correction. To calibrate the judge, we compare its decisions against human labels on the full set of 289 BrowseComp-ZH predictions. Treating the automatic judge's "equivalent" decision as the positive label and the human label as the reference, we measure approximately 99% F1 on the full set, where F1 is the harmonic mean of precision and recall for the binary answer-equivalence decision. The remaining disagreement is rare and falls into two main categories: false negatives, such as a gold answer containing a minor letter-level variant that the judge treats as a different answer, and false positives, such as a vague name abbreviation that the judge accepts although a human marks it as too imprecise.

Evaluation runs and score uncertainty. Unless otherwise noted, our scores average independent evaluation repeats (Anthropic, 2024): four runs for WideSearch, two for BrowseComp-ZH, and four for GAIA text-only. Answer generation uses temperature 1.0 and top-p 1.0. External baselines follow their reported settings. Our variants in Table 1 use the same models and asynchronous inference. For the WideSearch Item-F1 scores in Table 2, we instead report the mean $\bar{x}$ and sample standard deviation s over three independent evaluation runs with different random seeds. With the same RL-trained checkpoint, asynchronous and blocking inference achieve 68.2 ± 3.3 and 66.9 ± 2.3, respectively. We compute 95% Student-t confidence intervals as $\bar{x} \pm 4.303 s/\sqrt{3}$; they overlap. Table 4 uses the same three-run mean, standard deviation, and Student-t interval calculation for retrieval-based rollouts on GISA and WebQA.

Latency estimates and confidence intervals. For inference latency, we report the mean $\bar{\ell}$ and the sample standard deviation s in minutes per question across four timing measurements using the same RL-trained checkpoint. Asynchronous and blocking inference take 14.72 ± 1.44 and 23.81 ± 2.33 minutes, respectively. The 95% Student-t interval is $\bar{\ell} \pm t_{0.975,3} s/\sqrt{4}$ with $t_{0.975,3} \approx 3.182$, giving $[12.42, 17.01]$ and $[20.10, 27.52]$ minutes for the two modes, respectively.

For trace-retrieval training, we average all 237 completed GRPO step durations, including the initial steps and waiting for rollout data. To account for dependence between nearby steps, we use $10{,}000$ circular block-bootstrap resamples with blocks of 7 consecutive steps and random seed 2026. Each resample contains 237 durations. The 2.5th and 97.5th percentiles of the resampled means give an approximate 95% interval of $[5.14, 6.42]$ minutes for the mean latency of 5.73 minutes per step. The initial steps are slower; this interval describes the timing variation within the recorded run. For online RL, we retain the reported mean of 33.49 minutes and the 95% interval of $[29.21, 37.77]$ minutes, as step-level logs are not available. Table 2 reports training latency as the mean $\pm$ half the width of the interval (0.64 and 4.28 minutes). Bootstrap intervals can be asymmetric, so we report the lower and upper endpoints.

Paired significance tests. For Table 7, we randomly select one completed evaluation run per benchmark before computing behavioural statistics. Within that run, the initial and RL-trained swarms are evaluated on the same benchmark items ($n = 200$ for WideSearch and the full $n = 289$ set for BrowseComp-ZH), so we use paired two-sided approximate randomization tests rather than unpaired tests; paired designs for matched binary outcomes go back to the McNemar test (McNemar, 1947; Fagerland et al., 2013), and randomization-style tests are standard practice for comparing learning systems (Dietterich, 1998). For each benchmark and metric, we form item-level paired differences $d_i = m_i^{\mathrm{RL}} - m_i^{\mathrm{init}}$ and use the absolute mean difference $T = |n^{-1} \sum_i d_i|$ as the statistic. We then run 100,000 random sign-flip permutations of the paired differences and compute $p = (1 + \#\{T_b \geq T\})/(B + 1)$. Binary metrics use item-level $0/1$ values, while count and token metrics use their per-item scalar values. The reward row is the primary comparison. The remaining behaviour measurements were chosen before testing, and their stars report paired-test p-values without adjustment for multiple comparisons.

F Amortized Speedup Including Corpus Collection

We compare the expected total costs of online rollout generation and Trace-Retrieval Swarm RL, including one-time corpus collection. Let $S, K, G, E \in \mathbb{N}_{>0}$. The training set contains S questions. Corpus collection uses the open-source lead-agent model to generate K real-environment rollouts per question and stores the sub-prompt–final-response pairs in $\mathcal{D}$. Each of E GRPO epochs visits every question and generates G rollouts per question, where G is the GRPO group size.

For a complete rollout, let $C_1 > 0$ denote the mean generation cost of the lead agent, $C_2 > 0$ the mean cumulative online execution cost of the fixed sub-agents, and $C_3 \geq 0$ the mean cumulative cost of retrieving their final responses from $\mathcal{D}$. The sub-agent cost includes multi-turn reasoning and tool calls; our experiments use word Jaccard for retrieval. Define $\eta = C_2/C_1$ and $\delta = C_3/C_1$.

Cost model. We treat C_1, C_2, C_3 as fixed means, sharing C_1 across collection and both RL environments and C_2 across collection and online RL. Costs are additive in common resource units, with waiting excluded from C_1. We count one-time collection and rollout generation only, excluding indexing, reward computation, and GRPO updates. For time comparisons, we additionally assume that elapsed time equals cost divided by a common positive processing rate across these stages.

Theorem 1 (Total cost ratio including collection)**.** *Under this cost model, the ratio of expected total costs of online rollout generation to Trace-Retrieval Swarm RL, including collection, is*

$$\mathcal{A}(E, \eta, \delta) = \frac{GE(1+\eta)}{GE(1+\delta) + K(1+\eta)} = \left(\frac{1+\delta}{1+\eta} + \frac{K}{GE} \right)^{-1}. \tag{13}$$

Here K and G are fixed sampling settings. When $K = G$, this reduces to $\mathcal{A} = E(1+\eta)/[E(1+\delta) + 1 + \eta]$. For fixed K, G, η, δ, the ratio is strictly increasing in E and satisfies

$$\lim_{E \to \infty} \mathcal{A}(E, \eta, \delta) = \frac{1+\eta}{1+\delta}. \tag{14}$$

When $\delta = 0$, the limiting ratio is $1 + \eta$. With the assumed common processing rate, $\mathcal{A}$ also gives the ratio of expected elapsed times.

Proof. By linearity of expectation, the expected cost of the SGE online rollouts is

$$W_{\mathrm{real}} = SGE(C_1 + C_2). \tag{15}$$

Collecting SK real-environment rollouts once costs

$$W_{\mathrm{col}} = SK(C_1 + C_2). \tag{16}$$

Retrieval-based training generates SGE rollouts of mean cost $C_1 + C_3$, giving

$$W_{\mathrm{ret}} = SK(C_1 + C_2) + SGE(C_1 + C_3). \tag{17}$$

Taking the ratio and substituting $\eta = C_2/C_1$ and $\delta = C_3/C_1$ gives

$$\begin{aligned} \mathcal{A} = \frac{W_{\mathrm{real}}}{W_{\mathrm{ret}}} &= \frac{SGE(C_1 + C_2)}{SK(C_1 + C_2) + SGE(C_1 + C_3)} \\ &= \frac{GE(1+\eta)}{GE(1+\delta) + K(1+\eta)}. \end{aligned} \tag{18}$$

Equivalently, we can write the reciprocal of this ratio as

$$\frac{1}{\mathcal{A}} = \frac{1+\delta}{1+\eta} + \frac{K}{GE}. \tag{19}$$

For fixed K, G, η, δ, $K/(GE)$ strictly decreases to zero as E grows. Thus $\mathcal{A}$ strictly increases, with

$$\lim_{E\to\infty} \mathcal{A}(E,\eta,\delta) = \frac{1+\eta}{1+\delta}. \tag{20}$$

The cases $K = G$ and $\delta = 0$ follow by substitution. For expected elapsed times, dividing both costs by the assumed common processing rate leaves the ratio unchanged. □

Corollary 1 (Cost savings after two epochs)**.** *Under the same cost model, if $K \le G$, $\eta > 1$, and $0 \le \delta < (\eta-1)/2$, then*

$$\mathcal{A}(E,\eta,\delta) > 1 \qquad \textit{for every integer } E \ge 2. \tag{21}$$

The expected total cost is therefore lower even after corpus collection. With the assumed common processing rate, the same holds for expected elapsed time.

Proof. For $E \ge 2$, monotonicity from Theorem 1 and $K/G \le 1$ give

$$\begin{aligned} \mathcal{A}(E,\eta,\delta) \ge \mathcal{A}(2,\eta,\delta) &= \frac{2(1+\eta)}{2(1+\delta)+(K/G)(1+\eta)} \\ &\ge \frac{2(1+\eta)}{3+\eta+2\delta} = 1 + \frac{\eta-1-2\delta}{3+\eta+2\delta} > 1, \end{aligned} \tag{22}$$

where the final inequality follows from $\delta < (\eta-1)/2$.

Equivalently, the difference in total costs is

$$W_{\text{real}} - W_{\text{ret}} = SC_1\big[GE(\eta-\delta) - K(1+\eta)\big]. \tag{23}$$

Hence the exact condition for lower total cost is

$$GE(\eta-\delta) > K(1+\eta), \tag{24}$$

and the corollary provides sufficient conditions for this inequality. If both schemes additionally incur the same reward-computation and GRPO-update costs, adding them to each side preserves the strict inequality. The ratio in Theorem 1 applies to collection and rollout generation alone. □

Remark (Practical interpretation and reuse across experiments). We use $K = 4$ collection rollouts per question and a GRPO group size of $G = 8$, satisfying $K \le G$. The short retrieval latency reported in Appendix A supports treating δ as small for a fixed corpus and retrieval workload. The condition $\eta > 1$ requires cumulative online execution to cost more than generation by the lead agent. Model sizes, token counts, and tool-execution costs are needed to assess it; the number of sub-agent calls alone does not establish this condition. For settings satisfying these assumptions and $E \ge 2$, sufficiently small δ gives the cost savings in Corollary 1.

The analysis above considers a single RL experiment, whereas our corpus $\mathcal{D}$ is shared across RL experiments and ablations. If M experiments with the same sampling sizes, epoch count, and average cost parameters reuse one corpus, their combined cost ratio becomes

$$\mathcal{A}_M(E,\eta,\delta) = \left(\frac{1+\delta}{1+\eta} + \frac{K}{MGE}\right)^{-1}. \tag{25}$$

The one-time collection cost as a fraction of the online baseline cost decreases from $K/(GE)$ to $K/(MGE)$, further increasing the amortized speedup. When $K = G$, these fractions reduce to $1/E$ and $1/(ME)$, respectively.

G WIDE TRAINING DATA DIAGNOSTIC WITH MAX@K ITEM-F1

Table 5 compares real and retrieved sub-agent responses on wide training questions. Retrieved rollouts use the same 16,298-pair corpus $\mathcal{D}$ used for RL training, described in Appendix A. We sample four final answers per question under each condition, using real sub-agents or retrieved responses. Each answer is scored with the Item-F1 metric used for wide training data.

Max@k takes the highest Item-F1 among k sampled final answers for each question, then averages across questions. Let $s_{qi} \in [0, 1]$ denote the Item-F1 of sampled answer i for question q. For N questions, the reported score in percentage units is

$$M_k = \frac{100}{N} \sum_{q=1}^{N} \max_{1 \le i \le k} s_{qi}, \qquad k \in \{1, 2, 4\}. \tag{26}$$

The gap $\Delta = M_4 - M_1$ measures the gain from one to four rollouts in Item-F1 points. It is $73.2 - 57.2 = 16.0$ for real sub-agents and $72.0 - 56.3 = 15.7$ for retrieved responses. These diagnostics assess final-answer quality and the benefit of additional samples on wide training questions; Table 1 reports the separate held-out benchmark evaluation with real sub-agents.

H AI USE STATEMENT

Generative AI tools assisted with drafting and polishing selected passages and with formatting text, references, figures, and tables. They also supported retrieval and discovery by helping identify related work and locate relevant public technical documentation. The authors independently developed the research ideas and methods, designed and conducted the experiments, addressed technical problems, and analyzed and interpreted the results.