ASearcher: An Open-Source Large-Scale Reinforcement Learning Project for Search Agents

Jiaxuan Gao[12], **Wei Fu**[12], **Minyang Xie**[12], **Shusheng Xu**[2],
Chuyi He[2], **Zhiyu Mei**[2], **Banghua Zhu**[3], **Yi Wu**[12*]

[1] IIIS, Tsinghua University, [2] Ant Research, RL Lab
[3] University of Washington
samjia2000@gmail.com, jxwuyi@gmail.com

Abstract

Recent advancements in LLM-based agents have demonstrated remarkable capabilities in handling complex, knowledge-intensive tasks by integrating external tools. Among diverse choices of tools, search tools play a pivotal role in accessing vast external knowledge. However, open-source agents still fall short of achieving expert-level *Search Intelligence*, the ability to resolve ambiguous queries, generate precise searches, analyze results, and conduct thorough exploration. Existing approaches, including reinforcement learning (RL), supervised fine-tuning, and prompt-based techniques, face limitations in scalability, efficiency, and data quality. This paper introduces ASearcher, an open-source project for large-scale online RL training of search agents. Our key contributions include: (1) Scalable fully asynchronous RL training that maximizes GPU utilization, which decouples trajectory collection from model training to eliminate the GPU idle time while enabling long-horizon reasoning. (2) A prompt-based LLM agent that autonomously synthesizes high-quality, challenging, and uncertainty-aware QA pairs, creating a large-scale synthetic QA dataset for training. By applying large-scale training on high-quality synthetic data, ASearcher-Web-QwQ achieves Avg@4 scores of 42.1 on xBench-DeepSearch and 52.8 on GAIA, outperforming existing open-source search agents by a clear margin. Through RL training, 46.7% and 20.8% improvements are obtained on xBench-DeepSearch and GAIA, respectively. Notably, ASearcher-Web-QwQ exhibits extreme long-horizon search, with tool calls exceeding 40 times and generated tokens surpassing 150k during RL training. We open-source our models, training data, and codes.

1 Introduction

Recent advances in LLM-based agents have demonstrated remarkable capabilities in solving complex, knowledge-intensive problems by leveraging single or multiple external tools [42, 45, 37]. Among these, **search tools** stand out as particularly critical, enabling agents to access vast external knowledge for enhanced problem-solving [26, 8, 27]. However, expert-level use of search requires advanced intelligence. For instance, consider the question *"As of December 31, 2024, what were the numbers of gold, silver, and bronze medals won by China in the 2012 London Olympics?"*. A search agent must navigate noisy and conflicting answers from diverse sources, identify the root cause of conflicts as doping test disqualifications from official reports, and ultimately determine the correct information. Challenging real-world tasks require the agent to resolve high uncertainty in input queries, generate precise search queries, analyze and extract key insights from massive

* Corresponding author

Preprint.

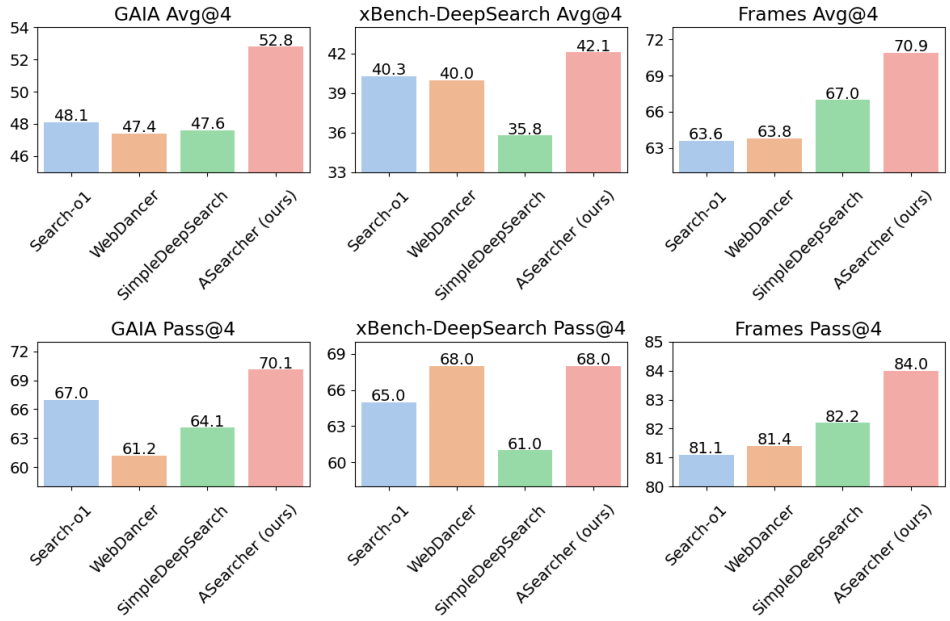


Figure 1: The performance of various methods based on 32B-scale models on GAIA, xBench-DeepSearch, and frames. Avg@4 and Pass@4 are reported.

data, resolve conflicting information, and conduct in-depth exploration. We term this sophisticated behavior *"Search Intelligence"*.

Proprietary agents and models has already exhibit signs of complex search behaviors through large-scale Reinforcement Learning (RL) training [1, 25]. However, open-source approaches for developing search agents still face significant limitations. A series of works employ *Reinforcement Learning* or *Supervised Fine-Tuning* approaches to incentivize tool-using capabilities [11, 30, 49, 33]. On the other hand, *prompt-based LLM agents* supported by open-source models could perform massive tool calls without training [18, 2]. However, in practice, we find that existing online RL approaches fail to incentivize complex and effective search strategies, and prompt-based LLM agents could fail due to model biases, such as easily committing to an incorrect answer. More recently, some works further build up on prompt-based LLM agents, utilizing offline RL approaches to improve the prompt-based agents [32, 19]. However, this offline RL paradigm, has been shown to underperform online RL in a broader range of domains [43, 6, 31].

In reasoning tasks such as math and coding, online RL has enable the models to evolve complex behaviors through iterative refining the reasoning processes based on correctness feedback. [9, 22, 7],. This raises a critical question: *How could online RL methods effectively unlock Search Intelligence in open-source agents?*

We identify two critical obstacles hindering effective online RL training for search agents:

- **Insufficient search turns limit complex strategy learning.** Existing works, such as Search-R1, artificially limit the number of search turns, e.g. ≤ 10 per trajectory, preventing the agent from exploring deeper search paths. However, complex queries often require multi-turn tool calls and multi-step reasoning, that could not be learned under strict turn limits.
- **Lack of large-scale, high-quality question-answer (QA) pairs:** RL training for reasoning tasks requires abundant, challenging, and correct QA pairs [3, 16, 46]. However, most existing open-source datasets for search agents are often outdated (e.g. HotpotQA), oversimplified, or too small , consequently failing to stimulate complex search behaviors through RL.

To address these challenges, we introduce ASearcher, an open-source project to enable *large-scale agentic RL training* for search agents. Our contributions include:

- **Long-horizon search via fully asynchronous agentic RL training.** With a large turn limit in batch generation RL training systems [11, 30, 21, 35], long trajectories within a batch could easily lead to significant idle time, slowing down the whole training process. Building up on AReaL [7], our fully asynchronous system avoids long trajectories from blocking the training by decoupling trajectory execution from model updates. This allows relaxed turn limits (e.g., 128 turns/trajectory), enabling agents to explore deeper search paths without sacrificing training efficiency. Remarkably, our agent, ASearcher-Web-QwQ, achieves extreme long-horizon search, with tool calls exceeding 40 turns and generated tokens surpassing 150k during RL training.
- **A scalable QA synthesis agent.** We design an LLM-based agent that autonomously generates **challenging, uncertain, and grounded** QA pairs requiring multi-turn tool use. Starting from seed questions, the agent iteratively *fuzzes queries* by obscuring key information, or *injects external facts* to increase complexity. Each constructed question undergoes *multi-stage validation* to ensure quality and difficulty. From 14k seed QAs, we generate 134k high-quality samples, with 25.6k requiring external tools for resolution.

Using ASearcher, we train agents equipped with search engines and browsers under two settings, *RL training starting from base models* (Qwen2.5-7B/14B), to demonstrate that our training pipeline incentivizes strong and generalizable search strategies, and *fine-tuning a prompt-based agent empowered by a powerful LRM (QwQ-32B)*, to validate the scalability of our training pipeline in fine-tuning large-scale prompt-based LLM agents.

We evaluate our agents with on multi-hop QA benchmarks and challenging benchmarks including GAIA [24] , xbench-DeepSearch [41], and Frames [14]. ASearcher-Local-7B/14B, trained only with local knowledge base, demonstrate surprisingly generalizability to realistic web search and achieve state-of-the art performances on multi-hop and single-hop QA tasks. Building up on QwQ-32B, ASearcher-Web-QwQ achieves an Avg@4 score of 42.1 on xBench-DeepSearch and 52.8 on GAIA, surpassing a set of open-source agents. When evaluating Pass@4, ASearcher-Web-QwQ achieves 70.1 on GAIA and 68.0 on xBench-DeepSearch. Notably, through RL training, ASearcher-Web-QwQ obtains 46.7% and 20.8% improvements on xBench-DeepSearch and GAIA, respectively.

To our knowledge, this is the **first large-scale open-source online agentic RL pipeline for LRM-based search agents**, unlocking Search Intelligence through scalable training and high-quality data. We hope our findings not only advance search agents but also inspire broader innovations in LLM agents for complex real-world tasks.

2 Preliminary

Multi-Turn Agentic Reinforcement Learning. We follow the formulation of Markov Decision Process (MDP). Formally, a MDP is defined by the tuple (S, A, T, R, γ). Here S represents the state space, usually containing the history, search results, and retrieved webpages. A denotes the action space and an action includes tokens generated by the agent. Some tool calling could be extracted from the action through specific tags, e.g. <search> search query </search>. $T(s'|s, a)$ is the transition probability, where s' is the updated state after applying the tool calling in action a at state s. Finally, $\gamma \in [0, 1)$ the discount factor and $R(s, a)$ is the reward function. Following prior practices [11, 30], we assume a sparse-reward setting where the reward is given at the end of the trajectory. In practice, we use rule-based functions or LLM-as-Judge as the reward function. At each timestep, the agent receives a state s_t and generates an action a_t with policy $\pi : S \to A$. The goal of the agent is to maximize the expected return,

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| a_t \sim \pi(s_t)\right] \tag{1}$$

3 Limitations of Existing Open-source Approaches

In this section, we provide a case study on an extreme challenging question from GAIA [24]. Specifically, we analyze Search-R1-32B [11] and Search-o1 (QwQ) [18] in Fig. 2.

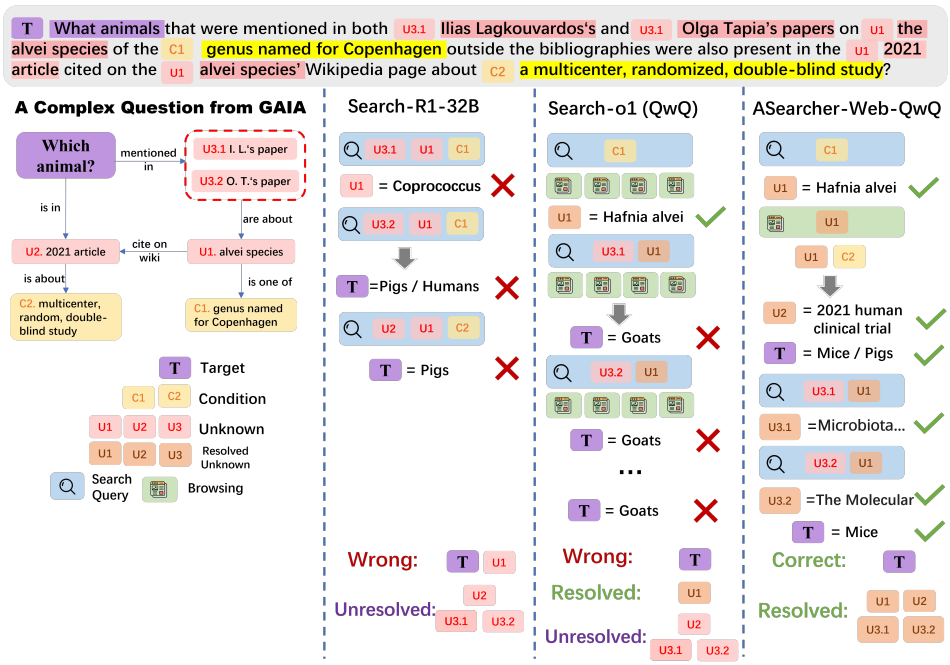


Figure 2: Case Study on a question from GAIA. This figure illustrates a complex query that involves 4 unknown variables and 2 conditions. **Search-R1-32B** makes redundant queries, is unable to extract key information, and performs unjustified reasoning, leading to an incorrect answer. **Search-o1 (QwQ)** can resolve one initial unknown variable U1 by making extensive webpage browsing, but could easily to commit to an incorrect answer. Our agent, **ASearcher-Web-QwQ**, is able to resolve all unknown variables and find the correct answer through precise queries, multi-turn tool calls, and step-by-step deduction.

Solution Path of the Sample Question. In Fig. 2, our case study is carried out on a question requiring finding some specific animal given 2 **constraints** and 4 **unknown variables**. To identify the correct answer, the search agent should first find out the mentioned species **U1** according to condition **C1**, identify the correct article **U2** that satisfies condition **C2**, and then find out the papers listed in **U3.1** and **U3.2**. Finally, the correct should be determined by cross reference the article **U2** and the papers **U3.1**&**U3.2**.

Existing Online RL Approaches Fail to Learn Complex Search Strategies. In Fig. 2, Search-R1-32B makes redundant queries, putting too many information in the search queries. Consequently, it obtains extreme noisy search results, and makes an incorrect resolution for the mentioned species **U1**. It also fails to identify the article **U2** and the papers **U3.1**&**U3.2**. This case study shows that existing online RL approaches only incentivize elementary search strategies.

Prompt-based LLM Agents Could Fail Due to Model Biases. In Fig. 2, Search-o1 (QwQ) can find the species name **U1** by an initial search on condition **C1** and browsing all related webpages. However, in the second query, though extensive webpage browsing are perform, it fails to resolve the correct paper in **U3.1** and makes an unjustified guess of the final answer. Consequently, though multiple following turns are performed, Search-o1 (QwQ) keeps misguided by the initial incorrect answer and fails to verify the correctness of the answer. Similar to Search-R1-32B, Search-o1 (QwQ) also fails to identify the article **U2** and the papers **U3.1**&**U3.2**.

4 ASearcher

In this work, we present ASearcher, an open-source project for unlocking search intelligence in search agents through large-scale RL training. As shown in Fig. 2, ASearcher trains a search agent that is able to solve complex questions by exhaustively resolving all uncertainties and performing

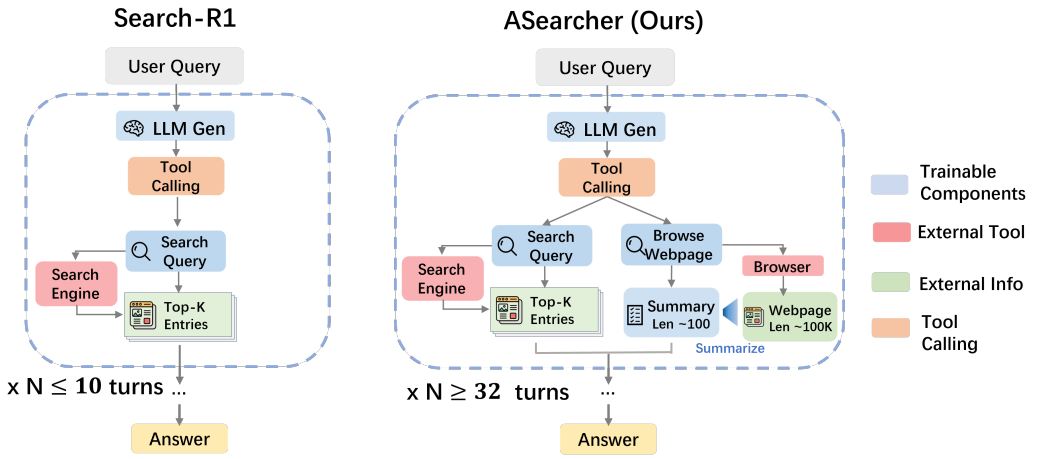


Figure 3: Comparison between ASearcher and Search-R1. (Left) Search-R1 is only equipped with search tools and lacks web browsing capability. (Right) ASearcher further incorporates web browsing tools. ASearcher is a comprehensive agent capable of both reasoning and processing lengthy web content. Notably, both reasoning and summarization abilities are optimized through RL training.

multi-turn tool calls. In the subsequent sections, we present the agent design, the training data as well as data synthesis agent, and fully asynchronous reinforcement learning training in ASearcher.

4.1 Agent Design

We employ a simple agent design in ASearcher, as illustrated in Fig. 3.

Tools. Given a user query, the agent can utilize two basic tools: **a search engine** and **a web browser**. The search engine takes a text query as input and returns relevant snippets along with their corresponding URLs. In The web browser accepts a URL and returns the content of the webpage. To effectively tackle complex problems, the model should strategically combine these tools and extract key information from the vast amount of data.

Webpage Summarization. Webpages could contain excessively long contents, therefore we employ the agent to summarize the webpage into a compact summary. At training time, this summarization process would also be optimized, allowing the agent to improve the summarization ability through RL training.

Instantiating ASearcher with Base LLMs and Advanced LRMs. Within the framework of ASearcher, we investigate two specific instantiations of the search agent: either *base LLMs* such as Qwen2.5-7B/14B, or *advanced Large Reasoning Models (LRMs)* such as QwQ-32B. These two different types of instantiations require different design choices in history management and prompting.

- For **base LLMs**, we following prior works [11, 30], to adopt *append-only* style prompting for the agent. Specifically, starting from a system prompt, all LLM-generated responses, search results and summaries of webpages are appended to the history. The agent takes as input the full history in chronological order and outputs some reasoning texts and actions. This approach ensures efficiency during inference time.
- For **LRMs**, LRMs are already equipped with instruction following capabilities. Therefore we follow the idea of Search-o1 [18] to instruct the LRM with different prompts for tool selection, summarization, and answering. We also note that LRMs typically generate long responses, and sometimes the history would be long. We need to ensure a compact input to ensure the LRM generates tokens with a sufficient budget. Therefore, in the history, we discard thinking processes but instead keep summarized thoughts and tool callings. When prompting the LRM, only the most recent 25k characters of the history are provided to the

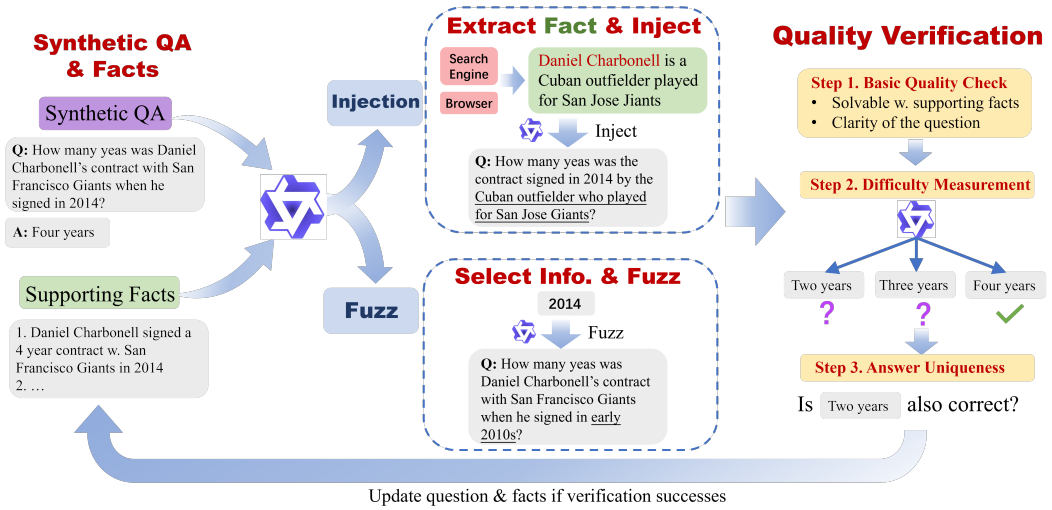


Figure 4: Data Synthesis Agent. Starting from a seed QA, the data synthesis agent iteratively modifies the question through two actions, *Injection* and *Fuzz*. Through *injection*, the agent enriches the question by adding some external facts. Through *Fuzz*, the agent blurs certain information to increase uncertainty and difficulty. The related fact to the question are tracked during the synthesis process. Each time the question is modified, a quality verification step is applied to ensure quality and difficulty of the synthetic questions.

LRM as additional context. These simple designs ensure that the LRM receives an input of at most 10k tokens.

End-to-End Reinforcement Learning. Finally, we highlight that the all LLM-generated responses of the agent, including the thinking process, tool calling, and summarization, are trained using Reinforcement Learning in an end-to-end manner.

4.2 Training Data

Our training data are from two primary sources. First, we carefully filter samples from open-source datasets to ensure difficulty and quality. Second, we synthesize high-quality question-answer (QA) pairs specifically designed to guide the agent to learn generalizable search strategies.

4.2.1 Open-source Data.

We begin with the training sets from HotpotQA[44] and 2WikiMultiHopQA[10], both of which are multi-hop QA datasets. We employ a model-based filtering process. We first train a model on the full set of open-source data with RL, and then generate 16 responses for each question using the trained model. Finally, we filter out questions that meat any of the following criteria,

- The model could not find a correct answer out of 16 responses
- The model achieves $\geq 50\%$ accuracy, meaning the question would not be challenging enough
- The model finds a correct answer with only a few search turns (i.e., ≤ 1 turns).

This filtering approach ensures we keep only the most challenging yet solvable questions that demand tool use. Finally, from a total of 304k QA pairs, we retain 16k challenging samples for RL training.

Additionally, we include a set of question-answer (QA) pairs designed for accessing certain webpages. In particular, we incorporate a small subset of WebWalkerQA[40] to help the model learn how to locate answers within noisy, real-world web search environments.

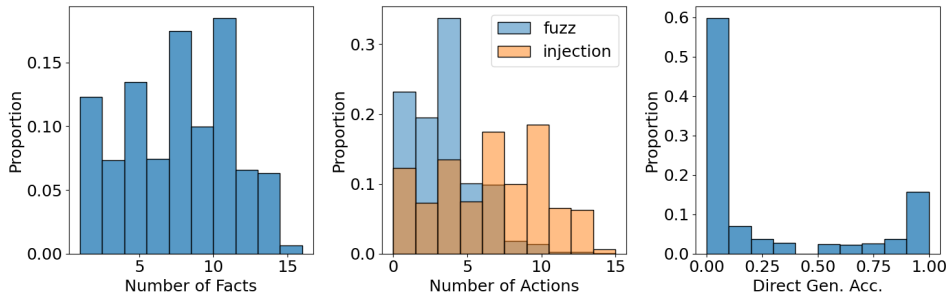


Figure 5: Statistics from our data synthesis process. (Left) The distribution of the number of supporting facts. (Middle) The distribution of the number of fuzz actions and injection actions. (Right) The accuracy distribution of QwQ-32B in answering the generated questions without using any tools.

4.2.2 Data Synthesis Agent

We further develop a data synthesis agent to create high-quality question-answer pairs. As shown in Fig. 4, the data synthesis agent begins with a seed question, and iteratively modifies the question to increase the complexity. To ensure the synthetic question is strictly aligned with reliable sources, a list of *supporting facts* obtained during the question synthesis process is kept and continuously updated for quality verification. At each step, given the current question and a list of supporting facts, the agent automatically selects between two key actions,

- **Action 1: Injection** aims to enrich the context of the question by inserting facts related to the question. The agent first selects an entity in the question and then obtains one piece of related fact about the selected entity from external sources such as Wikipedia. Then a new question is proposed by *injecting* the fact into the question. This injection action increases complexity of the question.
- **Action 2: Fuzzing** blurs certain details in the question to increase the uncertainty level of the question. For example, "Catskill Mountain Railroad" could be replaced with "a historic mountain railway". Through fuzzing the question multiple times, both the uncertainty level and difficulty of the question would gradually increase.

To ensure that a synthetic question is of high quality and to precisely evaluate the difficulty, we incorporate a rigorous *quality verification* phase for assessing synthetic questions,

- **Step 1. Basic Quality.** We employ an LLM to assess the basic quality of each question. This verification includes checking the clarity of the question and verifying whether the question-answer pair is accurate based on the supporting facts. This quality control step ensures that each question-answer pair is properly grounded in reliable sources.
- **Step 2. Difficulty Measurement.** We employ a cutting-edge LRM (e.g., QwQ-32B) to generate multiple answers directly for the synthetic question, without using any external tool. This verification process also serves as a measure of question difficulty.
- **Step 3. Answer Uniqueness.** The fuzzing action may loosen constraints excessively, compromising the uniqueness of the answer. To prevent ambiguity resulting from multiple correct answers, we evaluate whether any of the mismatched answers generated during the Difficulty Measurement step could serve as alternative valid answers.

We provide two illustrative examples in Tab. 1. Starting with a simple question, the injection action replaces specific entities with related factual details. For instance, “Michael P. Hein” is expanded to “who served as the first County Executive of Ulster County, New York...”. The fuzzing action introduces ambiguity by generalizing precise information, replacing the exact year “1934” with “the early 1930s” or substituting “Catskill Mountain Railroad” with “a historic mountain railway.”

Through iterative injection and fuzzing, the data synthesis agent produces questions that involve complex information and high uncertainty, requiring extensive search and reasoning to find the correct answer. After completing the question synthesis process, we filter out questions that the LRM can

Table 1: Examples of the synthetic questions, where red indicates injected facts and cyan represents fuzzed content.

Round	Action	Question
Seed QA	-	When was Michael P. Hein born?
Round 1	Injection	When was the Eckerd College alumnus who served as the first County Executive of Ulster County, New York, and graduated with a Bachelor of Arts in Business Administration born?
Round 2	Injection	When was the individual born who, as County Executive of Ulster County, New York, permitted the Catskill Mountain Railroad to continue operations between Kingston and Hurley during the 2016 United States House of Representatives elections and also held that position during the 2018 elections?
Round 3	Fuzzing	When was the individual born who, as County Executive of Ulster County, New York, permitted a historic mountain railway to continue operations between Kingston and Hurley during the 2016 United States House of Representatives elections and also held that position during the 2018 elections?
...	...	...
Round N	-	When was the first Ulster County Executive, known for disputes over the k and environmental issues related to the Ashokan Reservoir, who permitted the railroad's operations between Kingston and Hurley during the 2016 United States House of Representatives elections, born?
Seed QA	-	Where is the Riggs-Hamilton American Legion Post No. 20 located?
Round 1	Injection	Where is the American Legion Post in Russellville, Arkansas, built in 1934 and recognized as a notable example of WPA Rustic architecture and listed on the National Register of Historic Places located?
Round 2	Fuzzing	Where is the American Legion Post in Russellville, Arkansas, built in the early 1930s and recognized as a notable example of New Deal-era public works architecture and listed on the National Register of Historic Places located?
Round 3	Fuzzing	Where is the veterans' organization's building in Russellville, Arkansas, built in the early 1930s and recognized as a notable example of New Deal-era public works architecture and listed on the National Register of Historic Places located?
...	...	...
Round N	-	Where is the historic social hall and veterans' organization's building in Russellville, Arkansas, built during the 1930s and listed on the National Register of Historic Places in 1994 for its architectural significance as part of a federal jobs program during the Great Depression located?

directly generate the correct answer without relying on search tools. Since these questions can be answered solely based on the intrinsic knowledge of the model, they provide little value for enhancing search capabilities.

Starting with 14,107 seed questions, we perform an average of 6.3 injections and 3.2 fuzzes per question. From the synthetic pool, we select up to three high-quality variations per seed question. This curation process produces a final dataset of 25,624 entries, with the selected questions averaging 4.27 injections and 2.10 fuzzes each.

4.3 Asynchronous Agentic RL Training

4.3.1 Challenges of Scaling Up Trajectory Length in RL

In this section, we first empirically show that complex tasks require extensive tool calls and therefore RL training with a large turn limit is necessary for training advanced search agents. Then we show that variance of trajectory execution time is large during training, which could lead to significant idle time in batch generation RL systems.

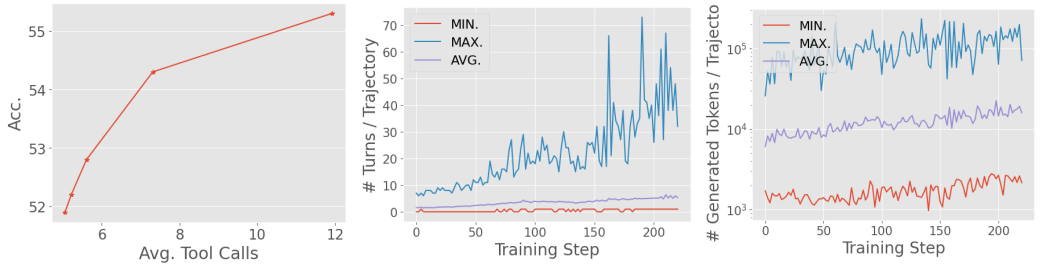


Figure 6: (Left) Test scaling of ASearcher-Web-QwQ. Data points are obtained by enforcing different minimum turns.The accuracy is averaged over GAIA, xBench-DeepSearch, and Frames. (Middle) Number of turns versus training steps. Long trajectories require much more turns than short trajectories. (Right) Number of generated tokens versus training steps. The number of output tokens exhibits significant variance, with long trajectories exceeding short ones by up to two orders of magnitude.

Complex Tasks Require Long Trajectories. Agentic tasks often require extensive LLM generations and multiple tool calls to solve complex problems, leading to prolonged trajectory execution time. As shown in Fig. 6(Left), we evaluate our RL-trained QwQ-32B agent on GAIA [24], xBench-Deepsearch [14] and Frames [14], requiring the agent to use tools for different numbers of turns. The results demonstrate that accuracy improves with more turns, confirming that complex tasks demand longer trajectories for effective problem-solving.

High Variance in Trajectory Execution Time. Long trajectories also introduce significant variance in execution time. We analyze turn counts and token generation during RL training of our QwQ agent (Fig. 6) and observe that the longest trajectories can span dozens more turns and two orders of magnitude more tokens than shorter ones. This disparity leads to highly unpredictable per-trajectory runtime, further complicating training efficiency.

Efficiency Issues of Agentic RL Training. Both prolonged execution and high runtime variance degrade RL training efficiency. We take one-step-off RL training system [21] as a representative example for batch generation RL systems. In one-step-off RL training, training for step N and trajectory generation for step N+1 are executed concurrently. As shown in Fig. 7, though this system overlaps trajectory rollouts with model training, batch generation remains bottlenecked by the slowest trajectory (e.g., trajectory 7), causing GPU idle time and under-utilization.

4.3.2 Fully Asynchronous RL Training.

To ensure efficient agentic RL training, we adopt a fully asynchronous training paradigm. Notably, our approach incorporates asynchornization at the two distinct aspects.

Asynchronous Trajectory Rollouts. Trajectory rollouts are collected in parallel and do not directly interfere with each other. Each trajectory independently sends tool calling requests to corresponding servers and LLM generation requests to the LLM inference engine. Concurrent requests from different trajectories are automatically handled by the servers. Fully independent trajectory execution ensures a trajectory does not need to wait for other trajectories when generating LLM responses and waiting for tool calling responses, thereby improving training efficiency.

Decoupled Rollout and Training. Besides asynchronous rollout, trajectory rollouts and model updates are also fully decoupled. In Fig. 7, we compare our fully asynchronous RL training with one-step-off RL training, which utilizes asynchronous rollout within batches. In fully asynchronous RL training, long trajectories do not block generation and can span multiple versions, significantly reducing GPU idle time and achieving near-full GPU utilization during generation. On the training side, a training step is launched as soon as sufficient trajectories are collected to form a batch. As shown in Fig. 7, the training process does not wait for the extremely long trajectory 7 but instead proceeds with trajectory 9.

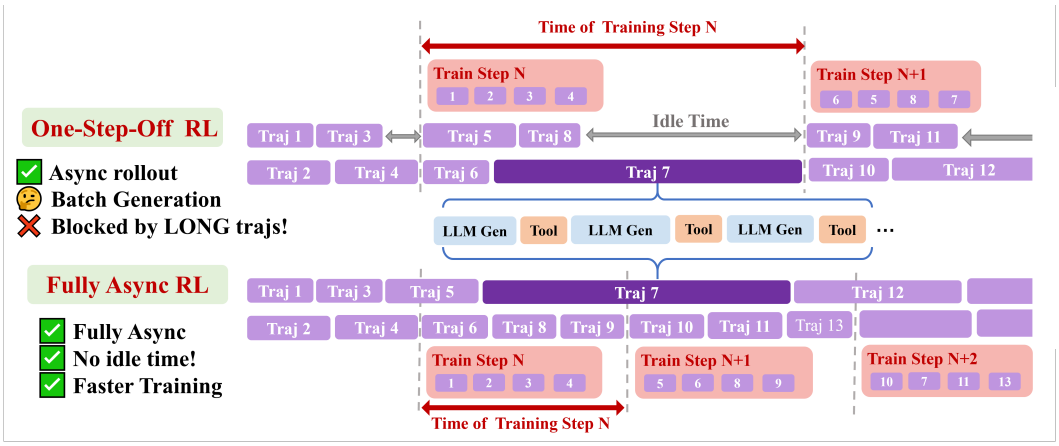


Figure 7: One-Step-off RL v.s. Fully Asynchronous RL. In batch generation systems, a batch should wait for the longest trajectory, leading to significant GPU idle time. In contrast, fully asynchronous RL achieves faster training than batch generation RL by fully decoupling training and trajectory generation, achieving near-full resource utilization for trajectory generation.

4.4 Training Details

GRPO Training. We employ the GRPO [29] algorithm to train the search agent. Specifically, for each input question x, G trajectories $\tau_1, \tau_2, \cdots, \tau_G$ are generated where $\tau_i = (s_0^i, a_0^i, s_1^i, \cdots, s_{T_i}^i)$. To optimize the agent, we employ the following loss,

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\{\tau_i\}_{i=1}^{G}\sim\pi_{\theta_{old}}(\cdot|x)} \left[\frac{1}{G}\sum_{i=1}^{G} \frac{1}{\sum_{t=0}^{T_i-1}|a_t^i|} \sum_{t=0}^{T_i-1}\sum_{j=1}^{|a_t^i|} \min\left(\frac{\pi_\theta(a_{t,j}^i|s_t, a_{t,<j}^i)}{\pi_{\theta_{old}}(a_{t,j}^i|s_t, a_{t,<j}^i)} \hat{A}_i, \right. \right. \tag{2}$$

$$\left. \left. \text{clip}\left(\frac{\pi_\theta(a_{t,j}^i|s_t, a_{t,<j}^i)}{\pi_{\theta_{old}}(a_{t,j}^i|s_t, a_{t,<j}^i)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_i \right) \right] \tag{3}$$

where ϵ is a hyperparameter, and $\hat{A}_i$ is the advantage for the i-th trajectory, computed based on the relative rewards of all trajectories within each group.

Dynamic Filtering. To enhance training efficiency, we implement dynamic filtering to exclude queries that lack meaningful training signals. Specifically, we remove queries where all responses yield identical rewards (resulting in zero advantages), including both queries where the agent already achieves high accuracy and those with incorrectly labeled answers.

Reward Function. For reward function, we adopt a sparse-reward setting where rewards are computed at trajectory completion. When training from base LLMs, the reward function combines a format reward and F1 score through multiplication. When fine-tuning LRM-based agents (e.g., QwQ), we utilize LLM-as-Judge[20][38] as the reward function and omit format rewards, as these models inherently maintain proper output formatting.

5 Experiments

5.1 Experiment Setup

Benchmarks. We first evaluate the agents on single-hop and multi-hop QA tasks. For single-hop questions, we use Natural Questions [15], TriviaQA [12] and PopQA [23]. For multi-hop questions, we use HotpotQA [44], 2WikiMultiHopQA [10], MuSiQue [36], and Bamboogle [28]. We further perform evaluation on more challenging benchmarks including Frames [14], GAIA [24], and xBench-DeepSearch [41] as extra test sets. We evaluate our approach on 1000 randomly sampled instances

Table 2: Results with Local Knowledge Base.

Method	Multi-Hop QA								Single-Hop QA						Avg.	
	2WikiMQA		HotpotQA		Bamboogle		Musique		NQ		TriviaQA		PopQA			
	F1	LasJ	F1	LasJ	F1	LasJ	F1	LasJ	F1	LasJ	F1	LasJ	F1	LasJ	F1	LasJ
7B Models																
Qwen-2.5-7B Direct Gen.	30.4	29.4	29.2	30.9	37.2	42.4	11.8	11.0	27.9	29.4	50.4	59.8	21.5	20.5	29.8	31.9
Search-R1-7B	54.7	58.1	57.6	60.8	**55.8**	58.4	28.2	27.1	**58.7**	49.9	68.0	78.0	57.3	55.7	54.3	55.4
R1-Searcher-7B	64.0	67.1	57.1	61.0	51.8	56.0	28.7	27.3	51.2	49.1	62.0	72.8	50.9	49.5	52.2	54.7
ASearcher-Local-7B	**72.3**	**77.6**	**62.6**	**67.6**	55.0	**60.0**	**34.4**	**32.6**	55.6	**54.5**	**68.1**	**79.3**	**57.9**	**55.9**	**58.0**	**61.0**
14B/32B Models																
QwQ-32B Direct Gen.	34.6	35.4	37.1	40.2	56.9	61.6	16.8	16.1	36.9	38.2	65.4	75.8	27.9	26.3	39.4	41.9
Search-R1-14B	48.2	49.8	56.2	58.9	52.8	51.2	27.0	25.7	60.0	51.2	71.0	79.9	56.1	54.3	53.0	53.0
Search-R1-32B	63.1	67.5	60.5	64.0	**60.0**	61.6	34.4	32.9	**60.8**	52.2	**72.0**	82.1	**60.3**	**58.2**	58.7	59.8
ASearcher-Local-14B	**72.2**	**79.1**	**65.1**	**71.0**	59.4	**64.8**	**35.6**	**34.6**	56.6	**56.1**	71.6	**84.0**	57.6	55.9	**59.7**	**63.6**

from the validation sets of HotpotQA, 2WikiMultiHopQA, and MuSiQue. For Bamboogle, Frames, GAIA and xBench-DeepSearch, we use their full test sets. For GAIA, we use the 103 examples from the text-only validation subset [18].

Search Tools. We evaluate the search agents with two settings, each with different types of search tools. In the first setting, **local knowledge base with RAG**, agents interact with a locally deployed RAG system to retrieve related information from a Wikipedia 2018 corpus [13]. In the other **web-based search and browsing** setting, agents operate in an interactive web environment with access to both a search engine and a browser tool. For more challenging benchmarks, GAIA, xBench-DeepSearch and Frames, we only conduct evaluations under this web-based setting.

Baselines We consider two groups of baselines aligned with the two benchmark categories. For the multi-hop and single-hop QA benchmarks, we include Search-R1(7B/14B/32B) [11], R1-Searcher(7B) [30], Search-o1(QwQ-32B) [18], DeepResearcher [49] and SimpleDeepSearcher [32]. We also prompt Qwen-2.5-7B/32B to directly generate answers without using any tools. On the more challenging benchmarks, we compare against powerful 32B-scale models, including direct generation with QwQ-32B, Search-o1(QwQ-32B) [18], Search-R1-32B [11], WebThinker-QwQ [19],SimpleDeepSearcher-QwQ [32] and WebDancer-32B [39]. All baselines are evaluated using the same tools as our agent to ensure a fair comparison.

Evaluation Metrics We adopt two complementary evaluation metrics: F1 score and LLM-as-Judge (LasJ). The F1 score is computed at the word level, measuring the harmonic mean of precision and recall between the predicted and reference answers. For LLM-as-Judge, a strong LLM (Qwen2.5-72B-Instruct) is prompted to assess the correctness of model outputs according to task-specific instructions. On GAIA, xBench-DeepSearch and Frames, we only use LLM-as-Judge and report the Avg@4 and Pass@4 scores for all models.

5.2 Main Results

We present the main experiment results across three evaluation settings: (1) local knowledge base with retrieval-augmented generation (RAG) on standard QA benchmarks, (2) web-based search and browsing on the same benchmarks, and (3) web-based search and browsing on more challenging benchmarks. **ASearcher**, instantiated with Qwen2.5-7B, Qwen2.5-14B, and QwQ-32B, consistently outperforms existing opensource agents of the same model scale on both F1 and LasJ metrics. ASearcher-14B achieves the best performance across 7B, 14B, and 32B models on a suite of multi-hop and single-hop QA benchmarks, and ASearcher-QwQ significantly outperforms several strong baselines of comparable size on these challenging benchmarks. These results highlight the generality and scalability of ASearcher across diverse tasks and model sizes.

Local Knowledge Base with RAG on Standard QA Benchmarks. As shown in Table 2, ASearcher-Local, trained via reinforcement learning with local knowledge base, achieves the best performance across 7B and 14B on a suite of multi-hop and single-hop QA benchmarks. In the 7B set-

Table 3: Results with Web-based Search and Browsing.

Method	Training Setting	Multi-Hop QA								Single-Hop QA						Avg.	
		2WikiMQA		HotpotQA		Bamboogle		Musique		NQ		TriviaQA		PopQA			
		F1	LasJ	F1	LasJ	F1	LasJ	F1	LasJ	F1	LasJ	F1	LasJ	F1	LasJ	F1	LasJ
7B Models																	
Qwen-2.5-7B Direct Gen.	-	30.8	30.9	28.6	29.5	37.2	39.6	10.6	1.9	29.6	29.9	51.2	59.3	19.8	17.4	29.7	29.8
Search-R1-7B	local	58.9	64.8	59.0	62.8	66.3	73.6	29.4	25.4	**58.4**	51.1	73.1	84.1	**53.0**	**51.3**	56.9	59.0
R1-Searcher-7B	local	66.6	69.4	56.8	61.6	62.8	72.0	28.7	25.3	49.6	48.7	67.6	79.5	46.5	45.2	54.1	57.4
DeepResearcher-7B	web	61.0	64.1	57.1	61.0	**68.8**	**76.8**	26.8	24.5	52.0	52.9	70.0	82.8	48.9	45.7	54.9	58.3
Simple DS-7B	web	67.4	73.9	57.6	62.5	61.5	72.0	26.4	26.2	43.9	53.1	73.9	85.4	43.7	48.8	53.5	60.3
ASearcher-Local-7B	local	**69.1**	**75.5**	61.6	67.1	66.2	76.0	**33.3**	**30.7**	54.7	**53.7**	**75.2**	**87.3**	52.9	49.7	**59.0**	**62.9**
ASearcher-Web-7B	web	67.5	73.3	**61.7**	**67.2**	66.4	72.0	32.9	29.6	55.2	55.4	74	85.7	52.4	48.9	58.6	61.7
14B/32B Models																	
QwQ-32B Direct Gen.	-	33.7	33.4	39.1	42.1	56.9	57.9	18.8	19.3	37.8	43.0	63.8	74.2	25.9	24.5	39.4	42.1
Search-o1 (QwQ-32B)	-	68.9	77.8	58.4	65.3	68.6	82.4	31.8	33.5	43.1	**57.2**	**76.3**	**89.6**	43.2	48.3	55.8	64.9
Search-R1-14B	local	51.8	53.8	55.3	58.6	67.4	75.2	29.8	26.9	57.7	49.6	74.4	83.9	51.0	49.8	55.4	56.8
Search-R1-32B	local	63.7	69.3	60.3	64.2	**76.4**	**81.6**	33.0	30.8	**58.6**	51.1	76.2	86.6	**55.0**	**53.6**	60.4	62.5
Simple DS-QwQ	web	71.7	80.4	62.0	67.5	73.2	83.2	33.3	32.9	45.7	55.3	77.2	90.2	45.5	47.8	58.4	65.3
ASearcher-Local-14B	local	70.4	79.8	**63.6**	70.5	68.7	80.8	35.1	33.8	53.5	55.4	76.1	88.5	52.5	50.5	60.0	**65.6**
ASearcher-Web-14B	web	**76.1**	**80.7**	63.5	**68.5**	69.9	75.2	**36.6**	**33.7**	56.0	55.5	75.4	87.6	52.9	50.0	**61.5**	64.5

Table 4: Results on GAIA, xBench-DeepSearch, and Frames. The results are evaluated with LLM-as-Judge. For baselines, we run the corresponding official codes for 4 seeds and report Avg@4 and Pass@4.

Method	GAIA		xBench-DeepSearch		Frames	
	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4
QwQ-32B Direct Gen.	23.1	31.1	11.8	23.0	29.9	39.9
Search-o1 (QwQ)	48.1	67.0	40.3	65.0	63.6	81.1
Search-R1-32B	28.6	43.7	19.5	37.0	44.1	61.0
WebThinker-QwQ	42.5	57.3	32.8	52.0	57.7	79.5
Simple DS-QwQ	47.6	64.1	35.8	61.0	67.0	82.2
WebDancer-QwQ	47.4	61.2	40.0	**68.0**	63.8	81.4
ASearcher-Web-QwQ	**52.8**	**70.1**	**42.1**	**68.0**	**70.9**	**84.0**

ting, ASearcher attains an average F1 of **58.0**, outperforming strong baselines such as Search-R1-7B (54.3) and R1-Searcher-7B (52.2). It also achieves a LasJ score of **61.0**, significantly outperforming Search-R1-7B (55.4) and R1-Searcher-7B (54.7). The gains are even more pronounced at the 14B scale, where ASearcher-Local-14B reaches an F1 of **60.0** and LasJ of **65.6**, surpassing even the larger 32B retrieval-based baseline Search-R1-32B.

Web-based Search and Browsing on Standard QA Benchmarks In Table 3, we evaluate agents in a realistic web-based setting.Notably, we evaluate models trained entirely with local knowledge base in the web setting in a zero-shot manner, to directly examine the generalizability of search strategies learned through RL. Across both model sizes, ASearcher consistently outperforms strong baselines. In particular, ASearcher-Web-14B achieves the best performance with an average F1 of **61.5**, surpassing SimpleDeepSearcher, the strongest 32B baseline in this setting. Remarkably, ASearcher-Local-14B model exhibits strong generalization when tested in the web-based setting, achieving significant gains over all baseline models of similar or larger size in terms of LasJ. This confirms that ASearcher learns generalizable search strategies that transfer to different sources of information.

Web-based Search and Browsing on Challenging Benchmarks. Table 4 shows experiment results on challenging QA tasks that require advanced problem-solving capabilities and search strategies. These benchmarks are specifically designed to assess the agent's ability to interact with real web and retrieve up-to-date information that often go beyond the internal knowledge of LLMs. As a result, direct generating answers from models (e.g., QwQ-32B) perform poorly across all datasets.Our agent, ASearcher-Web-QwQ, achieves the best Avg@4 scores on GAIA (52.8) and xBench-DeepSearch

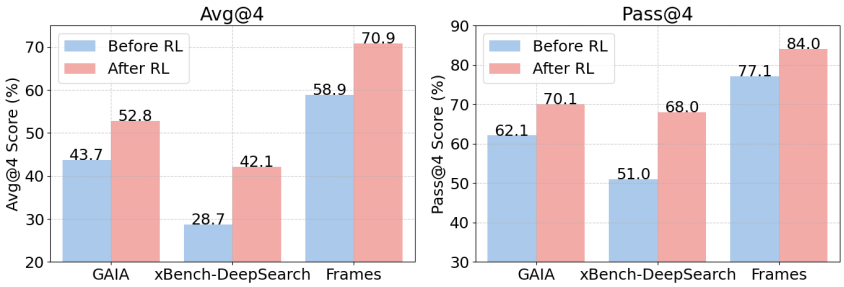


Figure 8: Comparison of the performance of QwQ-32B agent before and after RL Training.

(42.1), outperforming previous state-of-the-art open-source agents. These results further highlight superiority in handling long-horizon planning, real-world tool use, and open-domain exploration. Besides Avg@4, we also report the Pass@4 score that computes the ratio of questions that an agent finds the correct answer out of 4 trials. ASearcher-Web-QwQ also outperforms state-of-the-art open-source agents in terms of pass rate.

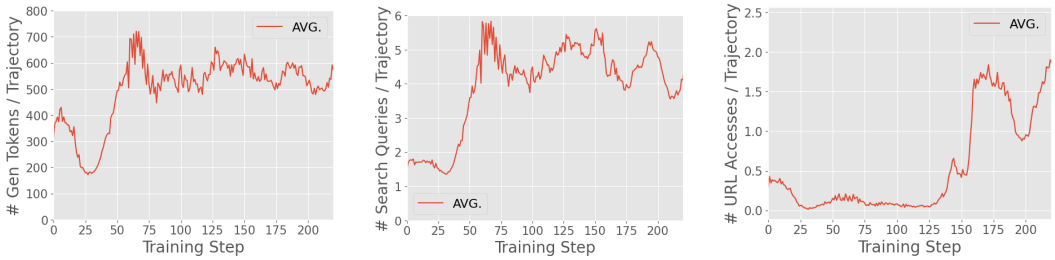
Effect of RL Training. As shown in Fig. 8, ASearcher-Web-QwQ obtains +9.1, +13.4, and +12.0 improvements on GAIA, xBench-DeepSearch and Frames respectively. When considering the pass rate, i.e. Pass@4, ASearcher-Web-QwQ also obtains significant gains, especially on xBench-DeepSearch with 17.0 improvements. Significant improvements in pass rate demonstrate that our training pipeline trains the agent to learn complex search strategies to perform precise searches, extract key information, and resolve conflict information.

5.3 Training Dynamics

(a) Generated Tokens (b) Search Queries (c) URL Accesses

Figure 9: Training Dynamics of ASearcher-Local-7B.

Training Dynamics of ASearcher-Local-7B/14B. In Fig. 9 and Fig. 10, we plot the number of generated tokens, search queries and webpage browsing for ASearcher-Local-7B and ASearcher-Local-14B training, respectively. With our training recipe, length increment and the increase in the



(a) Generated Tokens (b) Search Queries (c) URL Accesses

Figure 10: Training Dynamics of ASearcher-Local-14B.

number of tool callings are observed in both 7B and 14B scales. Notably, the number of search queries scale up to 6, which is higher than the numbers reported by prior works [11, 30]. Interestingly, we find that the 7B model fails to learn valid webpage browsing while the 14B model can learn to access webpage to solve challenging questions in the later stage of training. We hypothesize that the failure of 7B model in learning webpage browsing occurs because the model capacity is too small to stably learning summarize lengthy webpages in a zero RL training setting.

Training Dynamics of ASearcher-Web-QwQ. Similarly, the training dynamics of ASearcher-Web-QwQ are illustrated in Fig. 6. As the training progresses, the agent learns to perform more tool calls, reaching a maximum of around 40 calls at the 200th step, with peak instances even achieving up to 70 calls. Also the QwQ-32B agent generates more tokens through training, with a maximum of over 150k tokens. This scaling trend in both tool utilization and output length highlights the potential of fully asynchronous RL training for complex real-world agent applications.

6 Related Works

Search Agents. Some works have constructed agent workflows that enable large language models (LLMs) to leverage external tools for solving complex tasks, with notable examples include Search-o1[18] and ReAgent[48]. Prompt-based methods, while effective for rapid development, are fundamentally limited by the capacity of the underlying LLMs and could not be reliably improved with environment feedback. Some works attempt to construct SFT trajectories for LLMs. For instance, [4, 47] leverage large LLMs to synthesize retrieval and reasoning trajectories to fine-tune smaller models. Recently, some works investigate Reinforcement learning (RL) methods to enhance the LLM-based agents, mostly focusing on multi-hop QA benchmarks such as HotpotQA and 2WikiMultihop. [11, 30, 5, 49] perform RL training with multi-hop QA data and observe an increase in the number of tool uses. RAG-R1 [33] further combines SFT and RL to enhance the search strategies. More recently, researchers have begun to focus on more challenging tasks, by fine-tuning sophisticated prompt-based agents powered by Large Reasoning Models through offline RL [19], SFT on simulated trajectories with real-world web data [32, 17], and constructing challenging QAs for RL training. [34].

Synthetic Data for Search Agents. Rather than relying solely on large-scale human annotation, data synthesis has emerged as a scalable approach to prepare training data for search agents. Recent approaches generate synthetic but realistic QA trajectories by interacting with real web pages and curating data using LRMs [32, 39, 17]. On the other hand, WebSailor [17] constructs structurally challenging tasks through sampling and fuzzing, and WebShaper [34] utilizes techniques from set theory to construct high-quality complex QAs. By contrast, ASearcher develops an autonomous LLM agent for synthesizing challenging QAs with high uncertainty, without relying on complex knowledge graphs. Both the data synthesis agent and the synthetic training data in ASearcher are fully open-sourced.

7 Conclusion

In this work, we present ASearcher, a open-source project for large-scale RL training. Our contribution includes a fully asynchronous agentic RL training system and a data synthesis agent for large-scale high-quality QA construction. By instantiating ASearcher with base LLMs including Qwen2.5-7B/14B and prompt-based LLM agents based on QWQ-32B, ASearcher outperforms the state-of-the-art open-source agents across different model sizes and evaluation settings. With fully asynchronous agentic RL training and insight from our data synthesis pipeline, we hope our work could benefit future work on training advanced agents for a broader range of applications.

References

[1] Moonshot AI. Kimi-researcher: End-to-end rl training for emerging agentic capabilities. `https://moonshotai.github.io/Kimi-Researcher/`, 2025. URL `https://moonshotai.github.io/Kimi-Researcher/`.

[2] Salaheddin Alzubi, Creston Brooks, Purva Chiniya, Edoardo Contente, Chiara von Gerlach, Lucas Irwin, Yihan Jiang, Arda Kaz, Windsor Nguyen, Sewoong Oh, et al. Open deep search: Democratizing search with open-source reasoning agents. *arXiv preprint arXiv:2503.20201*, 2025.

[3] Chenxin An, Zhihui Xie, Xiaonan Li, Lei Li, Jun Zhang, Shansan Gong, Ming Zhong, Jingjing Xu, Xipeng Qiu, Mingxuan Wang, and Lingpeng Kong. Polaris: A post-training recipe for scaling reinforcement learning on advanced reasoning models. URL `https://hkunlp.github.io/blog/2025/Polaris`.

[4] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Self-reflective retrieval augmented generation. In *NeurIPS 2023 workshop on instruction tuning and instruction following*, 2023.

[5] Mingyang Chen, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z Pan, Wen Zhang, Huajun Chen, Fan Yang, et al. Learning to reason with search for llms via reinforcement learning. *arXiv preprint arXiv:2503.19470*, 2025.

[6] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2021. URL `https://arxiv.org/abs/2004.07219`.

[7] Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. Areal: A large-scale asynchronous reinforcement learning system for language reasoning, 2025. URL `https://arxiv.org/abs/2505.24298`.

[8] Google Team. Introducing Gemini deep research, 2025. URL `https://gemini.google/overview/deep-research/`. Accessed: 2025-04-06.

[9] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[10] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps, 2020. URL `https://arxiv.org/abs/2011.01060`.

[11] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

[12] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

[13] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pages 6769–6781, 2020.

[14] Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation, 2024. URL `https://arxiv.org/abs/2409.12941`.

[15] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Trans. Assoc. Comput. Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL `https://doi.org/10.1162/tacl_a_00276`.

[16] Jiazheng Li, Hong Lu, Kaiyue Wen, Zaiwen Yang, Jiaxuan Gao, Hongzhou Lin, Yi Wu, and Jingzhao Zhang. Questa: Expanding reasoning capacity in llms via question augmentation. *arXiv preprint arXiv:2507.13266*, 2025.

[17] Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, et al. Websailor: Navigating super-human reasoning for web agent. *arXiv preprint arXiv:2507.02592*, 2025.

[18] Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*, 2025.

[19] Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yutao Zhu, Yongkang Wu, Ji-Rong Wen, and Zhicheng Dou. Webthinker: Empowering large reasoning models with deep research capability. *arXiv preprint arXiv:2504.21776*, 2025.

[20] Yuxuan Liu, Tianchi Yang, Shaohan Huang, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. Calibrating llm-based evaluator. *arXiv preprint arXiv:2309.13308*, 2023.

[21] Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepcoder: A fully open-source 14b coder at o3-mini level. https://pretty-radio-b75.notion.site/DeepCoder-A-Fully-Open-Source-14B-Coder-at-O3-mini-Level-1cf81902c14680b3bee5eb349a512a51, 2025. Notion Blog.

[22] Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303 2025. Notion Blog.

[23] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Hannaneh Hajishirzi, and Daniel Khashabi. When not to trust language models: Investigating effectiveness and limitations of parametric and non-parametric memories. *arXiv preprint*, 2022.

[24] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

[25] OpenAI. Openai deep research. https://openai.com/index/introducing-deep-research/.

[26] OpenAI. Introducing deep research, 2025. URL https://openai.com/index/introducing-deep-research/. Accessed: 2025-04-06.

[27] Perplexity Team. Introducing Perplexity deep research, 2025. URL https://www.perplexity.ai/hub/blog/introducing-perplexity-deep-research. Accessed: 2025-04-06.

[28] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*, 2022.

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

[30] Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

[31] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for llm agents, 2024. URL https://arxiv.org/abs/2403.02502.

[32] Shuang Sun*, Huatong Song*, Yuhao Wang, Ruiyang Ren, Jinhao Jiang, Junjie Zhang, Lei Fang, Zhongyuan Wang, and Ji-Rong Wen Wayne Xin Zhao. Simpledeepsearcher: Deep information seeking via web-powered reasoning trajectory synthesis. 2025. URL https://github.com/RUCAIBox/SimpleDeepSearcher.

[33] Zhiwen Tan, Jiaming Huang, Qintong Wu, Hongxuan Zhang, Chenyi Zhuang, and Jinjie Gu. Rag-r1: Incentivize the search and reasoning capabilities of llms through multi-query parallelism. *arXiv preprint arXiv:2507.02962*, 2025.

[34] Zhengwei Tao, Jialong Wu, Wenbiao Yin, Junkai Zhang, Baixuan Li, Haiyang Shen, Kuan Li, Liwen Zhang, Xinyu Wang, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Webshaper: Agentically data synthesizing via information-seeking formalization, 2025. URL https://arxiv.org/abs/2507.15061.

[35] Prime Intellect Team, Sami Jaghouar, Justus Mattern, Jack Min Ong, Jannik Straube, Manveer Basra, Aaron Pazdera, Kushal Thaman, Matthew Di Ferrante, Felix Gabriel, et al. Intellect-2: A reasoning model trained through globally decentralized reinforcement learning. *arXiv preprint arXiv:2505.07291*, 2025.

[36] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

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

[38] Minzheng Wang, Longze Chen, Cheng Fu, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu, Nan Xu, Lei Zhang, Run Luo, et al. Leave no document behind: Benchmarking long-context llms with extended multi-doc qa. *arXiv preprint arXiv:2406.17419*, 2024.

[39] Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Gang Fu, Yong Jiang, et al. Webdancer: Towards autonomous information seeking agency. *arXiv preprint arXiv:2505.22648*, 2025.

[40] Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, et al. Webwalker: Benchmarking llms in web traversal. *arXiv preprint arXiv:2501.07572*, 2025.

[41] Xbench-Team. Xbench-deepsearch, 2025. URL https://xbench.org/agi/aisearch.

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

[43] Shusheng Xu, Wei Fu, Jiaxuan Gao, Wenjie Ye, Weilin Liu, Zhiyu Mei, Guangju Wang, Chao Yu, and Yi Wu. Is dpo superior to ppo for llm alignment? a comprehensive study, 2024. URL https://arxiv.org/abs/2404.10719.

[44] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

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

[46] 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*, 2025.

[47] Tian Yu, Shaolei Zhang, and Yang Feng. Auto-rag: Autonomous retrieval-augmented generation for large language models. *arXiv preprint arXiv:2411.19443*, 2024.

[48] Xinjie Zhao, Fan Gao, Xingyu Song, Yingjian Chen, Rui Yang, Yanran Fu, Yuyang Wang, Yusuke Iwasawa, Yutaka Matsuo, and Irene Li. Reagent: Reversible multi-agent reasoning for knowledge-enhanced multi-hop qa. *arXiv preprint arXiv:2503.06951*, 2025.

[49] Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*, 2025.