SAGE: Steerable Agentic Data Generation for Deep Search Agents with Execution Feedback

Fangyuan Xu[1*] **Rujun Han**[2] **Yanfei Chen**[2]
Zifeng Wang[2] **I-Hung Hsu**[2] **Jun Yan**[2] **Vishy Tirumalashetty**[2]
Eunsol Choi[1] **Tomas Pfister**[2] **Chen-Yu Lee**[2]
[1]New York University, [2]Google Cloud AI Research

Abstract

Deep search models, which aims to answer complicated questions requiring reasoning across multiple documents, can significantly speed up the information seeking process. Collecting human annotations for this application is prohibitively expensive due to long and complex exploration trajectories. We propose an agentic pipeline to automatically generate high-quality deep search question answer pairs for a given corpus and a target level of difficulty. Our pipeline, SAGE, consists of a data generator which generates QA pairs and an executor which attempts to solve the generated question, providing execution feedback for the data generator. The two components interact for multiple rounds to produce high quality (question, answer) pairs that satisfy the target difficulty level. Our intrinsic evaluation and analysis show SAGE generates queries that require diverse reasoning strategies, and significantly increases the correctness and difficulty of the generated data. Our extrinsic evaluation demonstrates >15% relative performance gain on wide range of popular deep search benchmarks, which highlights the effectiveness of SAGE in generating complexity queries and accurate answers.[1]

1 Introduction

Large language models (LLMs) are increasingly used as agents to interact with external environments and solve complicated tasks, such as coding (Jimenez et al., 2024; Dong et al., 2025), e-commerce and social forum discussion (Zhou et al., 2024; Peeters et al., 2025). Recently, there is a growing interest in building search-augmented agents that retrieve and reason about external information to solve complicated questions (Trivedi et al., 2023; Asai et al., 2024; Jin et al., 2025). High-quality, complex question–answer pairs are pivotal for training and evaluating capable search agents. Yet, such high quality data is not easily accessible and costly for human to annotate (Wei et al., 2025; Krishna et al., 2025).

Earlier work on retrieval-augmented generation (RAG) primarily focuses on questions where one search suffices to provide the necessary context (Joshi et al., 2017; Kwiatkowski et al., 2019; Han et al., 2024). Subsequent datasets (Yang et al., 2018; Trivedi et al., 2022) extended this setting to multi-hop reasoning. However, questions in these benchmarks typically require less than five retrieval and reasoning steps. Furthermore, their dependence on extensive human annotation or pre-existing structural information (e.g. inter-document links) makes such approach difficult to scale to tasks that demand longer reasoning and search chains, limiting their applicability.

To address the limitations of prior work, we propose an agentic data generation pipeline that leverages search-augmented LLMs to generate high-quality, challenging data for training and evaluating deep search agents. In contrast to most existing search-augmented QA datasets, which begin with a question and then retrieve supporting evidence to find the answer (Dunn et al., 2017; Kwiatkowski et al., 2019), our pipeline adopts reverse formulation to ensure the faithfulness of the generated data (Wei et al., 2025). Specifically, given a randomly sampled document from a corpus and a target difficulty level, our pipeline employs a search-augmented model to iteratively search and reason, generating an initial question–answer pair grounded in real retrieved evidence.

We find that reverse generation alone is not sufficient to consistently produce correct (question, answer) pairs that require a targeted number of search steps. As the target number of steps increase, the data generator model increasingly fail to generate

[*]Work performed during an internship at Google Cloud AI Research. Corresponding author: `fx2079@nyu.edu`, `{rujunh, chenyulee}@google.com`

[1]The code is released at `https://github.com/carriex/sage`.

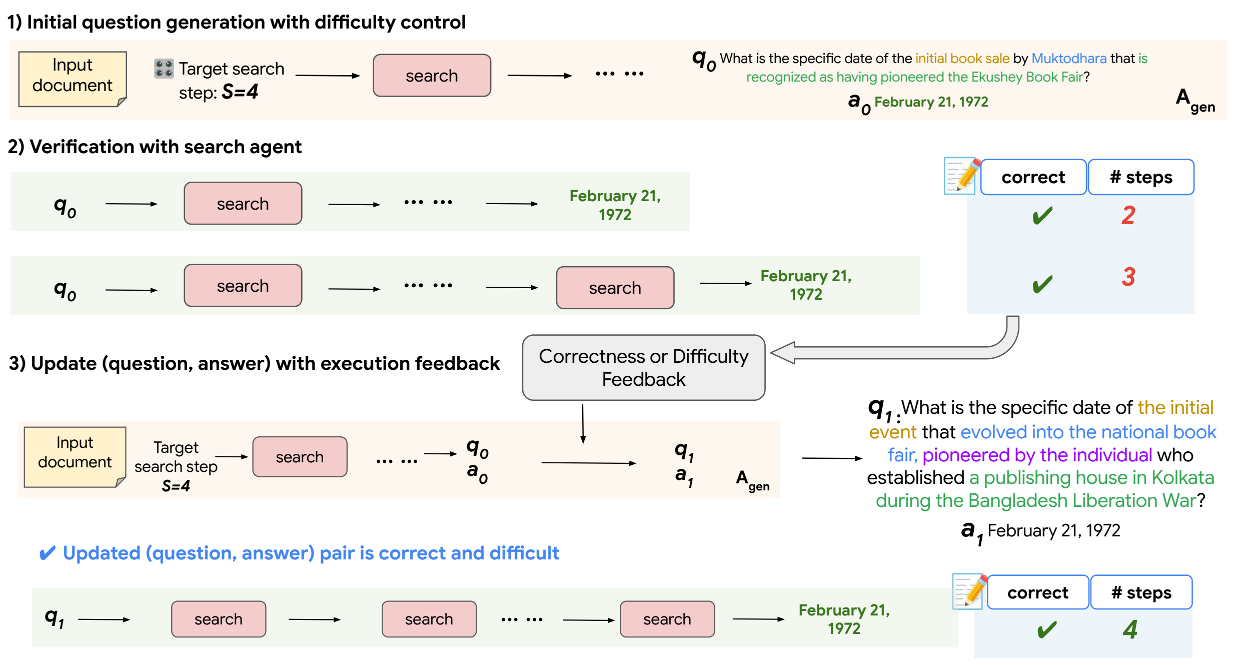
1) Initial question generation with difficulty control
Input document
Target search step: S=4
search
q0 What is the specific date of the initial book sale by Muktodhara that is recognized as having pioneered the Ekushey Book Fair?
a0 February 21, 1972
Agen
2) Verification with search agent
q0
search
February 21, 1972
correct
# steps
✔ 2
✔ 3
3) Update (question, answer) with execution feedback
Correctness or Difficulty Feedback
Input document
Target search step S=4
q0 a0
q1 a1
Agen
q1: What is the specific date of the initial event that evolved into the national book fair, pioneered by the individual who established a publishing house in Kolkata during the Bangladesh Liberation War?
a1 February 21, 1972
✔ Updated (question, answer) pair is correct and difficult
q1
✔ 4

Figure 1: Illustration of our proposed method: we first employ a data generator agent to generate an initial question, answer (QA) pair with a target number of search steps S for difficulty control. Our framework then leverages execution traces of the search agent as feedback for the data generator agent to generate correct and difficult data. In this example, the two initial QA pairs generated do not satisfy the target step S=4. After re-generation with search agent's feedback, the new QA pair becomes both correct and difficult. Noticeably, the question complexity increases from $\mathbf{q_0}$ to $\mathbf{q_1}$.

correct data that satisfy the target difficulty level. This reveals a common pitfall: a mismatch between the data generator's intended search plans and the actual number of search steps required to answer the question, as illustrated in Figure 1. Our observations inspire us to propose a dual-agent framework, which consists of a data generator agent and a search agent. The data generator agent first generates an initial draft of the question and answer pair. The generated (question, answer) pair is validated by a search agent to solve the question. We collect multiple traces from the search agent to gather feedback of the correctness and difficulty of the generated data. The *feedback* is fed into the data generator to produce a new QA pair.

Experiment results show that our framework, SAGE (**S**teerable **A**gentic **G**eneration with **E**xecution Feedback), significantly increases the quality of the generated data as measured by correctness and difficulty. We further conduct experiments on training search agents with our generated data, showing 30% improvements on in-domain evaluation set and up to 23% improvement for out-of-domain evaluation data across two model sizes (3B and 7B). Our analysis reveals that the generated data requires more diverse reasoning types than existing benchmarks.

2 Background

2.1 Search agent

Given an input question $\mathbf{q}$, a search agent $\mathcal{A}_{search}$ issues search queries $(s_0, s_1, ... s_i)$ to a retrieval tool $\mathcal{T}$ to gather information in a multi-turn manner before providing the final answer $\mathbf{a}$. Following the ReACT (Yao et al., 2023) framework, the agent alternates between outputting reasoning traces s_i and issuing search queries s_i, outputting a sequence of $\{r_0, s_0, r_1, s_1, ...r_i, s_i\}$. We omit the information returned by the search tool, which will be appended as input to the LM after s_i.

2.2 Training a search agent

While earlier work (Trivedi et al., 2023; Yao et al., 2023; Li et al., 2025b) explored prompting an LLM to use a search tool, recent work has shown that further training the model with either supervised learning (Asai et al., 2024) or reinforcement learning (Jin et al., 2025; Nakano et al., 2021) is beneficial. Supervised fine-tuning requires a set

Dataset	Annotation	# search ↑	Avg@8 ↓
	Large-scale training data		
NQ	Human	1.26	83.08
HotpotQA	Automatic	2.14	82.85
Musique	Automatic + human	2.67	64.44
	Small-scale evaluation data		
FRAMES	Human	3.19	74.28
Ours	Automatic	**4.85**	**79.54**

Table 1: Comparison with other multi-hop question answering datasets. To estimate dataset difficulty, we report both average number of searches (**# search**) and average performance of 8 samples from search-augmented gemini-2.5-flash.

of question $\mathbf{q}*$, answer $\mathbf{a}*$, as well as gold trajectory which consists of sub-queries, reasoning trace and information returned from search tool: $\{r_0, s_0, d_0, r_1, s_1, d_1 ... r_i, s_i, d_i\}$. As it is expensive to collect gold trajectories (Nakano et al., 2021; Asai et al., 2024), recent work (Jin et al., 2025) proposes training a search agent using reinforcement learning with outcome reward that measures the quality of the final answer $\mathbf{a}$ via exact match with a gold answer $\mathbf{a}*$. While training the model with reinforcement learning eliminates the need for gold trajectories, a set of high quality (q, a) pairs are essential for training a capable search agent.

2.3 Existing training and evaluation data

We review widely adopted datasets for training a search agent in Table 1. While there are large scale data annotated by human, they primarily consist of questions requiring a small number of steps to answer, as indicated by **# search** in Table 1. HotpotQA (Yang et al., 2018) and Musique (Trivedi et al., 2022) consist of multi-hop data, constructed through automatic or partially automatic pipelines. As constructing such a dataset can be cognitively heavy and time consuming for *human*, recent effort focuses on relatively small-scale human annotated evaluation set to benchmark performances of LLM agents (Krishna et al., 2025; Wei et al., 2025).

Concurrent work such as WebDancer (Wu et al., 2025) and WebShaper (Tao et al., 2025) propose automatic training data construction pipeline. They focus on the setting of using a browsing tool as the retriever, which is non-trivial to reproduce (Chen et al., 2025) and expensive to train due to excessive API costs. Moreover, their large-scale data is not yet publicly available. We provide a more comprehensive discussion of prior and concurrent efforts to synthetically generate deep search training data in Section 6.

3 Generating Challenging Synthetic Data for Search Agent: SAGE

Our goal is to generate (question, answer) pairs that require a search agent to issue multiple calls to a search tool and reason over the retrieved information before deriving the final (question, answer) pair. Our framework consists of the following components: a data generator agent $\mathcal{A}_{gen}$, a search agent $\mathcal{A}_{search}$, a retrieval model $\mathcal{T}$ and a corpus $\mathcal{D}$ containing a set of documents. In this section, we describe each component of our framework. The full procedure is outlined in Algorithm 1 and we include the implementation details such as prompt, decoding setting in Section A.1 in the appendix.

3.1 Initial data generation with difficulty prompt

Given an input document d randomly sampled from a corpus D, a data generator agent is instructed to generate an initial (question, answer) pair $\{q, a\}$ by iteratively issuing search query (s_0, s_1, ... s_i) interleaved with reasoning traces to collect relevant information from the corpus (Line 8 in Alg 1). The iterative process ends when the agent outputs an initial $\{q, a\}$ pair. The exact prompt used for this process is in Figure 4 in appendix. This is the reversed process of agentically solving a complicated question mirroring how human annotators typically construct such questions (Wei et al., 2025).

Difficulty prompt. Questions can have various levels of difficulty depending on the number of search and reasoning steps. We propose to use the number of steps S required by the search agent as a proxy measure for difficulty. Specifically, we include the target search step S in the input prompt to the data generator agent and instruct it to reason and plan in order to produce a question that requires the target number of search steps to solve.

3.2 Verifying the generated data

Generated data can be incorrect or does not satisfy the specified number of search steps. To verify the quality of the generated data, we use a search agent $\mathcal{A}_{search}$ to solve the generated question (Line 12 in Alg 1). Given the generated question q, the search agent iteratively issues search queries before producing an answer a'. We sample K traces from the search agent. We focus on two criteria for data quality:

Algorithm 1 Agentic Data Generation with Execution Feedback

```
Require: $\mathcal{A}_{gen}$: data generator agent; $\mathcal{A}_{search}$: search agent; $d$: input document; $S$: target search steps; $K$: number of search traces; $R$: max feedback rounds.
 1: $\mathcal{T}_{gen} \leftarrow \emptyset, \mathcal{T}_{search} \leftarrow \emptyset$        ▷ Initialize accumulated $\mathcal{A}_{gen}$ and $\mathcal{A}_{search}$ traces.
 2: IS_CORRECT $\leftarrow False$, IS_DIFFICULT $\leftarrow False$
 3: for $r \in \{0, \ldots, R\}$ do
 4:     if IS_CORRECT $\wedge$ IS_DIFFICULT then
 5:         break
 6:     else
 7:         if $r = 0$ then
 8:             $(q, a, t_{gen}) \leftarrow \mathcal{A}_{gen}(d, S)$        ▷ Generate initial (question, answer) pair
 9:         else
10:             $(q, a, t_{gen}) \leftarrow \mathcal{A}_{gen}(q, a, S, \mathcal{T}_{gen}, \mathcal{T}_{search})$   ▷ Re-generate the question and answer with execution feedback
11:         end if
12:         $(a^\star, S^\star, t_{search}$, IS_CORRECT, IS_DIFFICULT$) \leftarrow$ RunSearchAgent$(q, a, S, K, \mathcal{A}_{search})$   ▷ Collect execution.
13:         $\mathcal{T}_{gen} \leftarrow \mathcal{T}_{gen} \cup \{t_{gen}\}, \mathcal{T}_{search} \leftarrow \mathcal{T}_{search} \cup \{t_{search}\}$        ▷ Append generator and search traces.
14:     end if
15: end for
16: if IS_CORRECT then
17:     return $(q, a)$        ▷ Return correct pairs only.
18: end if
```

Algorithm 2 Verification with search agent

```
 1: function RunSearchAgent$(q, a, S, K, \mathcal{A}_{search})$
 2:     $\mathcal{T} \leftarrow [\,]$        ▷ All traces $(a'_k, S'_k)$
 3:     $\mathcal{M} \leftarrow [\,]$        ▷ Correct traces where $a'_k = a$
 4:     for $k \in \{1, \ldots, K\}$ do
 5:         $(a'_k, S'_k, t'_k) \leftarrow \mathcal{A}_{search}(q)$
 6:         append $(a'_k, S'_k, t'_k)$ to $\mathcal{T}$
 7:         append $(a'_k, S'_k, t'_k)$ to $\mathcal{M}$ if $a'_k = a$
 8:     end for
 9:     IS_CORRECT $\leftarrow (|\mathcal{M}| > 0)$
10:     if IS_CORRECT then
11:         $(a^\star, S^\star, t^\star) \leftarrow \arg\min_{(a', S', t') \in \mathcal{M}} |S'|$   ▷ Pick correct trace with least number of steps
12:     else
13:         $(a^\star, S^\star, t^\star) \leftarrow$ UniformSample$(\mathcal{T})$   ▷ Otherwise pick a random trace
14:     end if
15:     IS_DIFFICULT $\leftarrow (|S^\star| \geq S)$
16:     return $(a^\star, S^\star, t^\star$, IS_CORRECT, IS_DIFFICULT$)$
17: end function
```

- **Correctness:** which measures whether the generated (q, a) pair is correct. We treat pass@K performance of answer produced by the search agent (a') against the generated answer a as correctness, following previous work (Shi et al., 2025a).
- **Difficulty:** *minimal* number of search steps required among the correct traces from the search agent as an estimate of difficulty. If the number of search step is greater than or equal to the target search step S, we consider the generated data as difficult enough.

Algorithm 2 describes this verification process.

Generating complex (q, a) pairs requiring multiple search steps is non-trivial as it involves interaction with the retrieval tool. As revealed by our evaluation in Table 2, the data generator alone is only able to generate 18% of the data that satisfy both the correctness and difficulty constraint, when tasked to generate questions that require 3 to 7 steps to solve. We visualize the per-step performance in Figure 2, which further reveals that the data generator fails more frequently when attempting to generate questions requiring more search steps.

3.3 Generation with Execution Feedback

Failure to generate a correct (q, a) pair requiring the target number of search steps reveals a discrepancy between the data generator's trajectory and search agent's trajectory. For instance, two steps planned by the data generator might be solved with a single search step by the search agent, as we later analyze in Section 5.3. Can we leverage *both* trajectories to reconcile such discrepancy? Instead of merely leveraging search agent's execution results as a filter (Shi et al., 2025a), we propose to leverage the execution traces as *feedback* for the data generator.

Concretely, we feed both the data generator's traces and the search agent's traces, each containing the retrieved documents, back to the data generator. which is now tasked to output an updated (question, answer) pairs (Line 3-15 in Alg 1). This process can be conducted in an iterative manner, alternating between generating a new question and collecting execution feedback from the search agent for the updated (q, a) pair. Finally, we filter out (q, a) pairs for which none of the execution trace from the search agent arrives at the same answer as the data generator's answer (pass@K=0) .

System	Data Quality		Difficulty	
	% corr ↑	% pass↑	Avg@4↓	# search↑
	Baseline			
A_{gen} w/o S	84	-	86.25	3.16
A_{gen}	71	18	87.36	3.26
+1 resample	77	27	84.52	3.75
+2 resample	81	38	80.32	4.31
+3 resample	84	47	80.13	4.75
	Ours: SAGE			
+1 feedback	77	31	83.19	4.08
+2 feedback	83	42	80.38	4.60
+3 feedback	**87**	**50**	**79.54**	**4.85**

Table 2: Evaluation of data quality. We report the portion of correct (**% corr**) and portion of successful (**% pass**: data that are correct and require at least S steps to solve) generation out of all generated data. We measure difficulty of the correct portion of the generated data (pass@4=True) by Avg@4 and # search steps needed.

4 Experiments

We conduct intrinsic evaluation of our methods by measuring the correctness and difficulty of the generated data in Section 4.2. We further conduct downstream evaluation in Section 4.3 by training search agents on our generated data.

4.1 Experiment setting

Input corpus and retrieval setting. We use the 2018 Wikipedia dump as our input corpus (Karpukhin et al., 2020). Following (Jin et al., 2025), we use E5 (Wang et al., 2022) as the retriever module. The number of returned passages for each search call is set to 3.

Data generation pipeline setting. We use gemini-2.5-flash as the LLM which acts as both the data generator and the search agent. The maximum number of search steps are set to 20 for both the data generator model and the search agent. We report results with input target S set to 3 to 7 steps.

4.2 Intrinsic evaluation

Baselines. We compare our method with (1) **Data generator without difficulty prompt:** we prompt the data generator to generate a complicated question which requires multiple steps to solve without pre-specifying a target search step S; (2) **Initial data generator model:** we present the result of the initial data generator model; (3) **Resampling**: instead of using execution feedback to update the (question, answer) pair, we re-sampling another pair from A_{gen} for samples that are incorrect or not difficult enough. This is similar to best-of-K sampling with the search agent's execution result as the verifier. Note that both the resampling baseline and SAGE can be conducted in multiple rounds. We report results for up to 3 rounds.

Evaluation. We report various metric to measure the quality of the generated data: **% correct**: this reports the proportion of data out of all generated data that has pass@K=1, we set K as 4; **% pass**: this measures the proportion of data that is both correct and require the search agent at least S step to solve. Note that this metric is undefined for the baseline with no difficulty control. We further report two difficulty metrics for the questions that are correct: **Avg@4**: this measures the average performance of the search agent out of 4 traces. A lower **Avg@4** indicates a more difficult question. **Number of search steps:** the number of search steps required by the search agent to solve the question. We estimate this with the *minimal* number out of the correct traces. For correctness, we use LLM-as-a-judge against the answer proposed by the data generator as the reference answer.[2].

Results. Table 2 summarizes the results. Comparing A_{gen} with its variant without the target step S, we observe that explicitly including the target step in the prompt leads to a slight increase in the number of search steps required. However, A_{gen} alone struggles to generate questions that require the specified number of steps, achieving only an 18 % pass rate. Incorporating execution feedback from the search agent, either through the resampling baseline or execution feedback, substantially improves this success rate and yields more challenging questions overall. Among the two strategies, execution feedback consistently outperforms resampling across different numbers of rounds. Figure 2 further breaks down performance by target step, showing that feedback provides greater benefits than resampling as the target step increases, indicating its effectiveness for generating more complex questions. We include example questions generated by SAGE in Table 7 in the Appendix.

4.3 Downstream evaluation

We conduct extrinsic evaluation of data generated by SAGE by using it to train search agents. We adopt the Search-R1 (Jin et al., 2025) framework to train the agents and compare with agents trained with existing public data.

[2]We use `gemini-2.0-flash` as the LLM and the exact prompt is included in Table 6

Training Data	Backbone Model	In-domain						Out-of-domain	
		3-hop	4-hop	5-hop	6-hop	7-hop	AVG	Musique	FRAMES
-	gemini-2.0-flash	68.67	55.67	50.33	43.33	41.33	51.86	25.15	45.92
-	gemini-2.5-flash	80.00	67.00	57.33	48.67	37.33	58.07	28.03	50.04
NQ + HotpotQA	QWEN-3B	25.33	12.00	15.33	11.67	15.04	15.88	11.36	13.28
Musique	QWEN-3B	37.33	20.00	18.33	19.00	17.12	22.36	19.37	21.48
Ours	QWEN-3B	**42.33**	**26.67**	**25.33**	**25.00**	**23.29**	**28.52**	**19.93**	**23.83**
NQ + HotpotQA	QWEN-7B	45.00	26.33	25.67	24.67	23.63	29.06	18.93	26.17
Musique	QWEN-7B	48.33	28.67	24.67	25.00	21.23	29.57	21.59	25.00
Ours	QWEN-7B	**55.70**	**38.00**	**35.70**	**37.30**	**24.00**	**38.14**	**22.27**	**32.30**

Table 3: Downstream evaluation: performance of search agents on complex question answering datasets. We report performance using LLM as a judge against reference answers. Each of the data split contains 300 sampled data points. All models search for up to 10 turns.

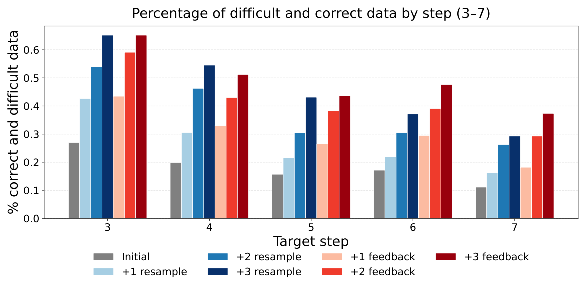

Figure 2: Percentage of correct and difficult data for target steps of 3 to 7 for various methods. Updating the question with execution feedback further improves the data quality compared to resampling, particularly with more target search steps.

Baselines. We consider models trained on publicly available question answering data: (1) **NQ+HotpotQA**: a combination of Natural Question (Kwiatkowski et al., 2019) and HotpotQA (Yang et al., 2018), which consists of 150K data. This is the training data used by Search-R1 (Jin et al., 2025); (2) **Musique** (Trivedi et al., 2022): consisting of multi-hop questions requiring two to four hops, the training split contains 20K data. We also compare with the performance of the prompting-based search agents using gemini-2.0-flash and gemini-2.5-flash as the agent.

Evaluation setting. We evaluate on various downstream question answering datasets that is grounded in Wikipedia: (1) **In-domain**: we report performance for testing data generated with SAGE. We generate 300 test data for each target search step and report the per-step performance as well as the average performance. We also consider two out-of-domain datasets: (1) **Musique**: which contains 2 to 4-hop questions; we randomly sample 300 data for each hop and report the average performance (2) **FRAMES** (Krishna et al., 2025): which contains multi-hop questions created by human annotators. We report the results on a randomly sampled set of 300 questions. We evaluate the performance with reference-based LLM-as-a-judge using `gemini-2.0-flash`, which outputs a binary judgment given the reference answer.

Data generation setting. We generate data using gemini-2.5-flash as both the data generator and search agent. Similar to Section 4.2, we use E5 as the retriever system and the 2018 Wikipedia dump as the corpus. For all dataset generation settings, we filter out questions that require less than two search steps and generate 20K data for training, such that the training set size is the same as our baselines. We report results for data generated with two rounds of feedback and conduct ablation studies on using data generated with different number of rounds in Section 5.1.

Model and training setting. We conduct experiment with two model variants: Qwen-2.5-3B-Instruct and Qwen-2.5-7B-Instruct (Yang et al., 2024). For **NQ+HotpotQA**, we directly inference with the Search-R1 checkpoints[3]. For models trained with musique and our data, we train the models using reinforcement learning with the reward defined as reference-based LLM-as-a-judge using `gemini-2.0-flash`. All models are trained with PPO (Schulman et al., 2017). Following prior work (Jin et al., 2025), we apply loss masking to the retrieved information. We include the implementation details in Section A.2 in the Appendix.

[3]`https://huggingface.co/PeterJinGo/SearchR1-nq_hotpotqa_train-qwen2.5-7b-it-em-ppo` and `PeterJinGo/SearchR1-nq_hotpotqa_train-qwen2.5-3b-em-ppo`.

Retrieval settings. For all training experiments, we use E5 as the retriever and the 2018 Wikipedia dump as the corpus, matching the set-up used for data generation. For inference, we use the same set-up except when evaluating on FRAMES, where we use the 2023 Wikipeida dump[4] to align with its data construction process (Krishna et al., 2025).

Results. Table 3 reports downstream performance of models trained on different QA datasets. Across both QWEN-3B and QWEN-7B, training on our generated data yields consistent gains over NQ + HotpotQA and Musique on in-domain evaluations. For QWEN-3B, training on our data increases average accuracy from 15.88% (NQ + HotpotQA) and 22.36% (Musique) to 28.52%. For QWEN-7B, while training on Musique provides only a marginal improvement over NQ + HotpotQA (29.57% vs. 29.06%), training on our data further boosts average accuracy to 38.14%. Training on our data also improves performance on out-of-domain datasets. On FRAMES, accuracy increases from 21.48% to 23.83% for QWEN-3B and from 26.17% to 32.30% for QWEN-7B. Notably, on Musique, QWEN-7B trained on our data achieves higher accuracy (22.27%) then directly on the in-domain Musique data (21.59 %).

4.4 Evaluation with Google Search

While our method generates challenging (q, a) pairs using a fixed corpus (e.g. Wikipedia), we further ask whether search agents trained with retrieval over a fixed corpus can generalize to other retrieval tool. To this end, we evaluate agents trained with Wikipedia-based retrieval on benchmarks that require Google search.

Datasets and settings. We evaluate on three benchmarks that require Google search: (1) **GAIA** (Mialon et al., 2023); (2) **Browsecomp** (Wei et al., 2025) and (3) **Humanity's Last Exam(HLE)-search** (Phan et al., 2025). For GAIA, we report results on the text-only subset with 103 questions. For Browsecomp, we evaluate on a randomly sampled subset of 200 questions. For HLE, we evaluate on a subset of questions that required search as classified by Gemini-1.5-pro, following prior work (Han et al., 2025). During inference, we replace Wikipedia-based retrieval with Google Search using the Serper API [5], retrieving the top three snippets per search query.

[4] `https://huggingface.co/datasets/wikimedia/wikipedia/viewer/20231101.en`

[5] `https://serper.dev/`

Training data	GAIA	Browsecomp	HLE-Search
	QWEN-3B		
NQ + HotpotQA	12.50	1.00	5.00
Musique	13.50	1.00	4.00
Ours	**18.80**	1.00	**5.50**
	QWEN-7B		
NQ + HotpotQA	14.60	1.60	4.50
Musique	15.60	2.10	**8.00**
Ours	**24.00**	**2.60**	7.00

Table 4: Results on deep search benchmarks using Google Search as the retrieval tool. We set the maximum number of search queries to 10 for GAIA and 20 for HLE and Browsecomp.

Results. Table 4 reports the results. We observe a promising trend indicating that training search agents with retrieval over a fixed corpus (e.g. Wikipedia) enables effective transfer when using a different retrieval tool (e.g. Google Search). Training on data generated by our pipeline leads to substantial improvements on GAIA compared to training on existing large-scale training data, for both 3B and 7B models (36% and 50% relative improvements compared to the strongest baseline respectively). We also observe improvement on Browsecomp, which consists of more multi-step search questions, for the QWEN-7B model. QWEN-3B achieves similarly poor performance (1.00% accuracy) across the three training datasets, likely because the questions are too challenging for a 3B model. On HLE, gains are more modest for both models, likely due to the substantial domain shift toward highly specialized scientific questions.

5 Analysis

5.1 Ablation on feedback rounds

In Section 4.3, we report downstream evaluation on training with data generated with SAGE with two rounds of feedback. How does the number of rounds impact downstream performance? We conduct an ablation study on training QWEN-7B model on data generated with 0-3 rounds of feedback. For all settings, we keep the training data size the same (20K) and filter out questions that require less than two search steps.

Results are reported in Table 5. We report downstream performance on the in-domain testing set, averaged across questions requiring 2-7 search steps, as well as on Musique and FRAMES. We

Round	Downstream Performance			Difficulty
	In-domain	Musique	FRAMES	Avg@4↓
0	33.62	18.70	29.00	86.25
1	33.62	19.49	29.30	83.19
2	**38.14**	**22.27**	**32.30**	80.38
3	34.14	20.89	28.13	79.54

Table 5: Ablation on training QWEN-7B on data generated with different numbers of feedback rounds.

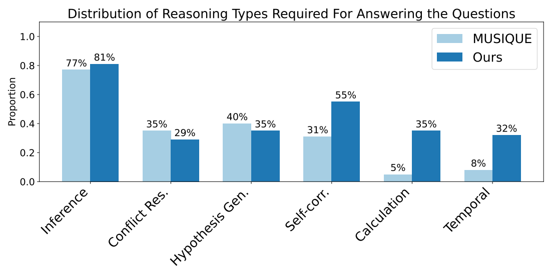

Figure 3: Distribution of reasoning types required to answer questions in Musique and our generated dataset. Our data spans a broad set of reasoning types, resulting in a more balanced distribution compared to Musique.

additionally report difficulty of the generated data, measured by **Avg@4**. We see improvement for both in-domain and out-of-domain datasets when increasing the number of feedback rounds from 0 to 2, confirming the effectiveness of incorporating execution feedback. Note that zero feedback round is equivalent to using the initial data generator only. We do not see further improvement when increasing the number of feedback to three rounds, despite that the data generated with three rounds of feedback being more difficult compared to those with two rounds. This suggests that increasing data difficulty alone is insufficient and highlights the need for more principled data curation strategies that balance difficulty and learnability.

5.2 Reasoning strategy analysis

Solving deep search questions require the agent to be able to issue search queries as well as *reason* about the retrieved documents. We conduct an analysis focusing on the reasoning strategies required to solve questions generated by our pipeline.

Setting. We collect search trajectory by prompting gemini-2.5-flash as the search agent. We sample 100 trajectories that lead to the correct answer for our generated data and Musique respectively. We then prompt gemini-2.5-flash to first (1) identify the types of reasoning strategies present in these search trajectories and (2) for each trajectory, label the presence of the reasoning strategies for each step. We include the prompt, definition for each strategy, as well as implementation details of the analysis in Section A.3 in the Appendix.

Category	Percentage
Easy data	
Superficial complexity	13%
Multi-query collapse	21%
Overly specific question	31%
Information co-location	35%
Incorrect data	
Ambiguous question	7%
Data generator error	19%
Search agent error	20%
Search agent retrieval failure	54%

Table 6: Error categorization for (q, a) pair generated by the initial data generator.

Results. Figure 3 shows the distribution of reasoning types required to solve questions in our dataset and Musique, note that a single question typically requires multiple strategies. SAGE produces questions that span a broader range of reasoning types, including conflict resolution, hypothesis generation, self-correction, calculation, and temporal reasoning. In particular, calculation and temporal reasoning are rare in Musique (5% and 8%, respectively) but appear more frequently in our data (35% and 32%), resulting in a more balanced distribution across reasoning categories.

5.3 Analyzing data generator's error

To better understand the main causes of the data generator's failure to produce correct and appropriately difficult (q, a) pairs, we analyze the discrepancies between the data generator's trajectories and the execution traces of the search agent. Specifically, we prompt gemini-2.5-flash to compare these two traces and categorize the error into error patterns. We identify four common failure patterns for the two error type (easy data and incorrect data). Results are reported in Table 6 and the prompt used for the analysis is in Section A.3 in the appendix.

For easy data, we observe that most errors stem from misalignment between the data generator's intended difficulty and the actual difficulty required by the search agent to solve the question, which is often influenced by the external environment. For instance, 35% of the generated question exhibit **information co-location**, where multiple pieces of information required to answer the question occurs in the same document in the corpus; 21% of the generated data exhibits **multi-query collapse**, in which information from *different documents* can be retrieved by the retrieval tool using a single query. Both phenomena reduce the number of search steps required to answer the question. These analysis motivates our method as such discrepancy is only discoverable via execution feedback.

For incorrect data, the misalignment between the search agent's answer and the data generator's answer most commonly arises from failure on the search agent, including retrieval failure or reasoning errors. While we do not distinguish these data from truly incorrect data and filter them out for training, future work could explore verifying the correctness of (q, a) pairs unsolved by the search agent. Around 20% of the incorrect data is attributable to errors made by the data generator, such as hallucinating ungrounded steps in the question generation process. The remaining 7% of data involves ambiguous question which leads to different answer found by the search agent.

6 Related Work

Deep Search. Early retrieval-augmented generation (RAG) systems typically perform a single retrieval step before generating an answer (Lewis et al., 2020; Gao et al., 2023). Recent agentic RAG approches aim to support deep search, in which models iteratively interleave retrieval and reasoning to solve questions requiring multiple search steps. Trivedi et al. (2023) is among the earlier work that explore a prompting-only strategy that guides model to alternate between search and reasoning until reaching an answer. Search-o1 (Li et al., 2025c) furthre improves deep search by incorporating in-document reasoning to filter out irrelevant retrieved information. Search-R1 (Jin et al., 2025) proposes training search agents following the ReACT framework (Yao et al., 2023) using outcome-based reward. Subsequent works (Shi et al., 2025b; Lu et al., 2025; Wu et al., 2025) all adopt RL training paradigm as high-quality SFT trajectories for deep search agent is difficult to collect at scale. We also adopt the Search-R1 training framework for our downstream evaluation.

Synthetic data generation for deep search. We review several concurrent work on synthetic data generation for deep search. WebPuzzle (Shi et al., 2025b) adopts a similar reverse QA generation process and difficulty filtering as in our work, using both open-domain web pages and Wikipedia. WebDancer (Wu et al., 2025) proposes two datasets: one by browsing web-pages and then produce different types of QA pairs, one by selecting an existing QA pair to iteratively refine the question by searching about an extracted entity in it. DeepDive (Lu et al., 2025) leverages random walk over a knowledge graph with LLM-based obfuscation, to improve correctness and difficulty of generated data, while WebShaper (Tao et al., 2025) proposes to use set formalization of entity relations to expand query complexity. WebSailor (Li et al., 2025a) and WebExplorer (Liu et al., 2025) generate challenging questions around pre-selected rare entities through progressive obscuration. In contrast, our approach relies on strong LLM-based data generator and search agents to flexibly synthesize challenging data, avoiding dependence on high quality knowledge graph and structure imposed by explicit entity relations. We further note that several concurrent works rely on commercial retrieval API such as Google Search during data generation. While Google Search covers more domain, it substantially increases the cost for both data generation and model training due to the large number of intermediate search queries required. Our pipeline is grounded in a fixed corpus while experiments in Section 4.4 demonstrate promising transfer to use Google Search at inference time only.

Finally, existing baseline training data Natural Question (Kwiatkowski et al., 2019), HotpotQA (Yang et al., 2018) and Musique (Trivedi et al., 2022) are constructed either through human annotation, automatic pipeline leveraging Wikipedia structure or a combination of both. To our best knowledge, these are the only other publicly available training data for search agents grounded in Wikipedia.

7 Conclusion

We introduce an agentic pipeline, `SAGE`, to automatically generate deep search data for a given corpus. Our framework consists of a data generator agent that generates complicated question as well as answers, and a search agent that provides execution feedback by attempting to solve the question. We conduct comprehensive evaluation on both intrinsic data quality and downstream evaluation, training search agent's with our generated data. Evaluation on in-domain and out-of-domain evaluation set demonstrates the effectiveness of our framework.

Limitations

Proposed method. Our dual-agent framework currently relies on a fixed search agent to provide execution feedback for the data generator agent to evolve the generated data. Future work can explore *co-evolving* both agents (e.g., through iterative training), which could further enhance the quality of the generated data and hence the capability of the search agent. We adopt pass@K=1 as the correctness criterion for the generated data, which serves as a practical approximation but may still admit hallucinated or incorrect content. Future work could investigate more robust verification methods for identifying correct (q, a) pairs that the existing search agent cannot solve (pass@K=0).

Experiment setting While we evaluate our method using both intrinsic evaluation and downstream evaluation, our experiments do not explore alternative reinforcement learning algorithms such as GRPO (Shao et al., 2024), nor model scales beyond 7B parameters. We only experiment with generating data from a single general domain corpus, Wikipedia. Future work can explore generating deep search data for other domain-specific corpus, such as legal or scientific domains.

Acknowledgments

We thank Tengxiao Liu, Siru Ouyang, Yihe Deng, Rui Meng and other members from Google Cloud AI Research for helpful discussions and feedbacks throughout the project.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. 2025. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *ArXiv*, abs/2508.06600.

Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025. A survey on code generation with llm-based agents. *ArXiv*, abs/2508.00083.

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Guney, Volkan Cirik, and Kyunghyun Cho. 2017. Searchqa: A new qa dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *ArXiv*, abs/2312.10997.

Rujun Han, Yanfei Chen, Zoey CuiZhu, Lesly Miculicich, Guan Sun, Yuanjun Bi, Weiming Wen, Hui Wan, Chunfeng Wen, Solène Maître, George Lee, Vishy Tirumalashetty, Emily Xue, Zizhao Zhang, Salem Haykal, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. 2025. Deep researcher with test-time diffusion. *Preprint*, arXiv:2507.16075.

Rujun Han, Yuhao Zhang, Peng Qi, Yumo Xu, Jenyuan Wang, Lan Liu, William Yang Wang, Bonan Min, and Vittorio Castelli. 2024. Rag-qa arena: Evaluating domain robustness for long-form retrieval augmented question answering. *arXiv preprint arXiv:2407.13998*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. 2025. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4745–4759, Albuquerque, New Mexico. Association for Computational Linguistics.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur 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. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025a. Websailor: Navigating super-human reasoning for web agent. *Preprint*, arXiv:2507.02592.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025b. Search-o1: Agentic search-enhanced large reasoning models. *ArXiv*, abs/2501.05366.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025c. Search-o1: Agentic search-enhanced large reasoning models. *CoRR*, abs/2501.05366.

Junteng Liu, Yunji Li, Chi Zhang, Jingyang Li, Aili Chen, Ke Ji, Weiyu Cheng, Zijia Wu, Chengyu Du, Qidi Xu, Jiayuan Song, Zhengmao Zhu, Wenhu Chen, Pengyu Zhao, and Junxian He. 2025. Webexplorer: Explore and evolve for training long-horizon web agents. *Preprint*, arXiv:2509.06501.

Rui Lu, Zhenyu Hou, Zihan Wang, Hanchen Zhang, Xiao Liu, Yujiang Li, Shi Feng, Jie Tang, and Yuxiao Dong. 2025. Deepdive: Advancing deep search agents with knowledge graphs and multi-turn rl. *arXiv preprint arXiv:2509.10446*.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. *Preprint*, arXiv:2311.12983.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Ouyang Long, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. Webgpt: Browser-assisted question-answering with human feedback. *ArXiv*, abs/2112.09332.

Ralph Peeters, Aaron Steiner, Luca Schwarz, Julian Yuya Caspary, and Christian Bizer. 2025. Webmall – a multi-shop benchmark for evaluating web agents. *ArXiv*, abs/2508.13024.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, Adam Khoja, Ryan Kim, Richard Ren, Jason Hausenloy, Oliver Zhang, Mantas Mazeika, and 1093 others. 2025. Humanity's last exam. *Preprint*, arXiv:2501.14249.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Jun-Mei Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *ArXiv*, abs/2402.03300.

Wenxuan Shi, Haochen Tan, Chuqiao Kuang, Xiaoguang Li, Xiaozhe Ren, Chen Zhang, Hanting Chen, Yasheng Wang, Lifeng Shang, Fisher Yu, and Yunhe Wang. 2025a. Pangu deepdiver: Adaptive search intensity scaling via open-web reinforcement learning. *ArXiv*, abs/2505.24332.

Wenxuan Shi, Haochen Tan, Chuqiao Kuang, Xiaoguang Li, Xiaozhe Ren, Chen Zhang, Hanting Chen, Yasheng Wang, Lifeng Shang, Fisher Yu, and Yunhe Wang. 2025b. Pangu deepdiver: Adaptive search intensity scaling via open-web reinforcement learning. *arXiv preprint arXiv:2505.24332*.

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. 2025. Webshaper: Agentically data synthesizing via information-seeking formalization.

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

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *ArXiv*, abs/2212.03533.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Is abella Fulford, Hyung Won Chung, Alexandre Passos, William Fedus, and Amelia Glaese. 2025. Browsecomp: A simple yet challenging benchmark for browsing agents. *ArXiv*, abs/2504.12516.

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025. Webdancer: Towards autonomous information seeking agency.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, and 25 others. 2024. Qwen2.5 technical report. *ArXiv*, abs/2412.15115.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*.

Figure 4: Prompt used for initial data generator.

Prompt used for initial data generator.

Your task is to generate a complicated question that will require a search agent `target_search_step` search steps to answer by gathering information using a search engine.
You will first reason about the initial document and plan for gathering comprehensive information inside <think> and </think>.
You will then call a search engine by <search> query </search> and it will return the top searched results between <information> and </information> to collect information.
You must conduct reasoning inside <think> and </think> first every time you get new information.
You will call the search engine for `n_search_step` steps. After `n_search_step` searches, you must provide the question inside <question> and </question>, the answer inside <answer> and </answer>, and the answering step inside <answering steps> and </answering steps>. You can use your own knowledge to construct the search query, but the final answer and each of the answering step must be supported by the information you gathered from the search engine.
The question should be understandable standalone as the agent will use the question to search for information without access to the initial document.
An example question: How much did the film in which Jake Gyllenhaal played his second lead role gross in its initial run at the box office?
Avoid How and Why question.
The answer should be `answer_type` and short.
Make sure the answer is correct and **unique** for the question generated.
Initial document: `context`

A Appendix

A.1 Implementation details for SAGE

For data generator and search agent, we set temperature to 1 and disable thinking for gemini-2.5-flash. If the data generator does not generate a (q, a) pair after issuing the maximum number of search calls, we force it to generate a (q, a) pair by appending "`<think>I have used up all the search budget and I will use the existing information to formulate a new plan and generate the question, answer, and answering plans.`" to the prompt.

Data generator setting We include the prompt used for the initial data generator agent in Table 4; and the prompt for feedback in Figure 7 and Figure 8.

Search agent setting We include the prompt used for search agent in Table 5.

LLM-as-a-judge setting We include the prompt used for reference-based LLM-as-a-judge in Prompt 6. We use gemini-2.0-flash with temperature 0.

A.2 Implementation details for training the search agent

PPO objective. We conduct downstream evaluation by training search agents with the Proximal Policy Optimization (PPO)(Schulman et al., 2017). It optimizes the language model by maximizing the below objective in Figure 9, with π_{theta} and π_{old} representing the current the previous policy model; $I(y_t)$ representing whether the loss masking apply to the token, we apply loss masking to retrieved document ($I(y_t) = 0$). ε is a clipping hyperparameter, and the advantage estimate A_t is computed using Generalized Advantage Estimation (Schulman et al., 2015).

Figure 5: Prompt used for search agent.

Prompt used for search agent.

Answer the given question by using a search engine.
You will first reason about the question inside <think> and </think>, for instance, break down the question into multiple sub-questions that you will search for.
You must call a search engine by <search> query </search> and it will return the top searched results between <information> and </information>.
Try to formulate the search query in the form of a question.
After receiving the information, you must reason about it inside <think> and </think> before issuing a new query or providing the final answer.
Each of your reasoning step should be grounded in the retrieved information. Do not use your own knowledge, but you can use commonsense knowledge or arithmetic knowledge.
Do not use your own knowledge to write the query, the query should be based on the question and the retrieved documents.
Do not infer the entities in the question, but you can use the entities in the retrieved documents to write the query.
You can search as many times as your want. Try to break down the question for each search query and gather comprehensive information.
If you have gathered enough information to answer the question, you can provide the answer to the query inside <answer> and </answer>, without detailed illustrations.
Generate an answer based on the retrieved information, instead of your own knowledge.
This is an example answer: <answer>Beijing</answer>. Question: `question`

Training details. For training the search agent, we set the learning rate of the policy LLM to 1e-6 and that of the value LLM to 1e-5. Training is conducted for 500 steps, with warm-up ratios of 0.285 and 0.015 for the policy and value models, respectively. We use Generalized Advantage Estimation (GAE) with parameters $\lambda = 1$ and $\gamma = 1$. Training is conducted on a single node with 8 H100 GPUs. We use a total batch size of 512, with a mini-batch size of 256 and a micro-batch size of 64. The maximum sequence length is set to 8,192 tokens, with a maximum response length of 1024 and a maximum length of 1000 tokens for retrieved content. We set the maximum number of retrieval calls to 8.

A.3 Analysis details

We include the prompt for reasoning strategy analysis in Prompt 10, which also presents the definition for each of the category.

Figure 6: Prompt used for reference-based LLM-as-a-judge.

Prompt used for reference-based LLM-as-a-judge.

Judge whether the following [response] to [question] is correct or not based on the precise and unambiguous [correct_answer_list] below. Each answer in the [correct_answer_list] is separated by a comma.
[question]: `question`
[response]: `model_answer`
Your judgment must be in the format and criteria specified below:
extracted_final_answer: The final exact answer extracted from the [response]. Put the extracted answer as 'None' if there is no exact, final answer to extract from the response.
[correct_answer_list]: `gold_answer`
reasoning: Explain why the extracted_final_answer is correct or incorrect based on [correct_answer_list], focusing only on if there are meaningful differences between answer in the [correct_answer_list] and the extracted_final_answer. Focus on recall, i.e. if the extracted_final_answer covers all the points in the answer in the [correct_answer_list]. It is ok if it provides more details. It is also ok if the extracted_final_answer misses minor point from the correct_answer, as long as it is evident that they are referring to the same thing. Do not comment on any background to the problem, do not attempt to solve the problem, do not argue for any answer different than [correct_answer_list], focus only on whether the answers match. Ignore capitalization.
correct: Answer 'yes' if extracted_final_answer matches any of the answers in [correct_answer_list] given above, or is within a small margin of error for numerical problems. Answer 'no' otherwise, i.e. if there is any inconsistency, ambiguity, non-equivalency, or if the extracted answer is incorrect.
confidence: The extracted confidence score between 0% and 100% from [response]. Put 100 if there is no confidence score available.

Example questions
Previous question: On what date did Tottenham Hotspur win their first major trophy, a feat that made them the only non-league club to achieve it, while under the chairmanship of the individual who took office the year before the club's establishment at its original ground, colloquially known as "The Lane"?   *Previous answer:* April 27, 1901   *Updated question:* What was the date of the replay match in which the player, who notably scored goals in every round of Tottenham Hotspur's first major trophy victory, helped his team secure that trophy after the club had become a limited company under a chairman who served until 1943?   *Updated answer:* April 27, 1901
Previous question: What is the death date of the actor who sang songs in the 1944 film that was Naushad's first musical success?   *Previous answer:* 2 August 1979   *Updated question:* What is the death date of the actor, born in Gujranwala, who sang songs in Naushad's first musical success film of 1944?   *Updated answer:* 2 August 1979
Previous question: What was the precise date Sir Frank Whittle, a key figure honored at the Midlands Air Museum, received his knighthood as Knight Commander of the Order of the British Empire (KBE) for his pioneering work on the jet engine?   *Previous answer:* July 1, 1948   *Updated question:* What was the precise date of the military retirement of the figure whose heritage center is located within the museum adjacent to the former Electric Railway Museum site?   *Updated answer:* 26 August 1948

Table 7: Example question generated and updated by our pipeline.

Figure 7: Prompt used for incorporating execution feedback to update the incorrect (question, answer) pair.

Prompt used for incorporating execution feedback to update the incorrect (question, answer) pair.

You will be given an output from a question generator agent, which generates a complicated question, answer pair; as well as the output from a search agent, which attempts to solve the question generated in a fixed number of turns.
The answer from the search agent is not the same as the data generator agent. You task is to examine their traces and output the correct question, answer pair based on their retrieved documents. You can update either the question, the answer or both.
You will first reason about why is there a discrepancy between the search agent's answer and the data generator's answer. Output your reasoning trace inside <reason> and </reason>. You will then reason about how to update the question answer pair to make sure it is correct and requires the agent `target_step` search step to answer. A search step is defined as a call to the search tool. Output your reasoning trace inside <think> and </think>. For factual information, you should ONLY rely on the context provided for the data generator agent and the documents retrieved by both the data generator and search agent (inside <information> and </information>).
If you find it non-trivial to update just the question and answer, you can generate a new question answer pair ONLY based on the retrieved documents.
The updated question should require the search agent at least `target_step` search steps to answer. However, the answer should be short, such as an entity, a date or a number. The question should be understandable standalone, as the search agent will solve the question without access to the documents (they will need to search for them).
When you are ready to provide the new question, answer pair, you can provide the question inside <question> and </question>, the answer inside <answer> and </answer>, and the search step inside <search steps> and </search steps>. For each search step, output the exact search question; the sub-answer to the search question; and the retrieved document from the search agent and data generator agent's output that supports the sub-answer. Make sure each step is absolutely needed to answer the question and there is no short cut. Tip: use retrieved document from different steps so avoid two sub-queries being solved by one search query.
# Data generator agent
Prompt: `data_generator_agent_prompt`
Agent's output: `data_generator_agent_response`
# Search agent
Prompt: `search_agent_prompt`
Agent's output: `search_agent_response`
# Your output

Figure 8: Prompt used for incorporating execution feedback to update the easy (question, answer) pair.

Prompt used for incorporating execution feedback to update the easy (question, answer) pair.

You will be given an output from a question generator agent, which generates a complicated question, answer pair to be solved by a search agent for at least `target_step` **search** steps; as well as the output from a search agent, which attempts to solve the question generated. The search agent is able to solve the question in less than`target_step` search steps. Your task is to update the question so that it requires the search agent more steps to solve.

You will first reason about why the search agent is able to solve the question in fewer steps. Output your reasoning trace inside <reason> and </reason>. You will then reason about how to update the question so that it will require more search steps. For factual information, you should ONLY rely on the context provided for the data generator agent and the documents retrieved by both the data generator and search agent (inside <information> and </information>), without relying on other information not in the retrieved context. Output your reasoning trace inside <think> and </think>. If you find it non-trivial to update the plan, you can generate a new question answer pair ONLY based on the retrieved documents.

The updated question should require the search agent at least `target_step` search steps to answer. Note that some of the answering steps do not involve search and thus do not count. However, the answer should be short, such as an entity, a date or a number. The question should be understandable standalone, as the agent will solve the question without access to the documents (they will need to search for them).

When you are ready to provide the new question, answer pair, you can provide the question inside <question> and </question>, the answer inside <answer> and </answer>, and the search step inside <search steps> and </search steps>. For each search step, output the exact search question; the sub-answer to the search question; and the retrieved document from the search agent and data generator agent's output that supports the sub-answer. Make sure each step is absolutely needed to answer the question and there is no short cut. Tip: use retrieved document from different steps so avoid two sub-queries being solved by one search query.

# Data generator agent

Prompt: `data_generator_agent_prompt`

Agent's output: `data_generator_agent_response`

# Search agent

Prompt: `search_agent_prompt`

Agent's output: `search_agent_response`

# Your output

$$\mathcal{J}_{\text{PPO}}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_{\text{old}}(\cdot|x;\mathcal{R})}\left[\frac{1}{\sum_{t=1}^{|y|} I(y_t)} \sum_{t=1:I(y_t)=1}^{|y|} \min\Big(\frac{\pi_\theta(y_t \mid x, y_{<t};\mathcal{R})}{\pi_{\text{old}}(y_t \mid x, y_{<t};\mathcal{R})} A_t,\ \text{clip}\big(r_t(\theta), 1-\varepsilon, 1+\varepsilon\big) A_t\Big)\right]$$

Figure 9: PPO objective for training search agents.

Figure 10: Prompt used for reasoning strategy analysis.

Prompt used for reasoning strategy analysis.

You will be given a question and a trace of a search agent solving this question. You will analyze and categorize the behavior for each of the thinking step.
Below are some example categories, please feel free to propose new ones as you see appropriate:
- Information inference: the agent makes an inference based on the piece of information in the retrieved document in its reasoning.
- Conflict resolution: the agent reasons about conflicting information in the documents and makes a decision.
- Calculation: The agent performs numerical calculation.
- Temporal reasoning: The agent performs temporal reasoning, such as deriving duration between two dates.
- Self-correction: The agent recognizes that the previous search failed to yield the required information (a list of stations). It re-evaluates its state, confirms its goal, and decides to try a more specific search query.
- Hypothesis Generation: The agent makes a guess / hypothesis that is not grounded in the retrieved documents.

Output format: only return the list of strategies for each step:
- Step i: [list of strategies]
- Step i+1: [list of strategies]

Question: `question` Agent's reasoning trace: `agent_trace`