# EvolveSearch: An Iterative Self-Evolving Search Agent

Anonymous ACL submission


## Abstract

The rapid advancement of large language models (LLMs) has transformed the landscape of agentic information seeking capabilities through the integration of tools such as search engines and web browsers. However, current mainstream approaches for enabling LLM web search proficiency face significant challenges: supervised fine-tuning struggles with data production in open-search domains, while RL converges quickly, limiting their data utilization efficiency. To address these issues, we propose **EvolveSearch**, a novel iterative self-evolution framework that combines SFT and RL to enhance agentic web search capabilities without any external human-annotated reasoning data. Extensive experiments on seven multi-hop question-answering (MHQA) benchmarks demonstrate that EvolveSearch consistently improves performance across iterations, ultimately achieving an average improvement of 4.7% over the current state-of-the-art across seven benchmarks, opening the door to self-evolution agentic capabilities in open web search domains.


## 1 Introduction

Rapid advances in large language models (LLMs) have enabled agentic AI capabilities through tool integration (*e.g.*, search, browsing, code execution), supporting autonomous interaction with external environments. Recent agentic systems like OpenAI *Deep Research* (OpenAI, 2025) achieve 51.9% accuracy on BrowseComp, well above human performance (29.8%) (Wei et al., 2025), highlighting LLMs' potential for deep information research.

Existing agentic systems are primarily implemented via prompting-based, supervised fine-tuning (SFT)-based, and reinforcement learning (RL)-based approaches. Prompting-based agents rely on predefined workflows (Anthropic, 2025; Zhou et al., 2023), resulting in rigid behaviors and limited generalization. They often struggle with instruction following and reasoning, requiring substantial manual prompt engineering for reliable performance (Pan et al., 2025). Subsequent work distills agentic capabilities into smaller LLMs via supervised fine-tuning (SFT) (Wang et al., 2025; Wu et al., 2025a). However, in open-ended search tasks, collecting SFT data necessitates complex environment interactions, making data construction challenging. More importantly, they lack robustness in complex, real-world environments (Zheng et al., 2025). RL-based approaches (Shao et al., 2024) have recently gained attention for enabling models to acquire decision-making capabilities through online interactions with the environment and reward-driven updates. This paradigm allows agents to adapt to task dynamics in an end-to-end manner. However, in practice, existing RL-based methods often converge within fewer than 100 steps, resulting in low data efficiency and limited performance gains (Song et al., 2025; Jin et al., 2025).

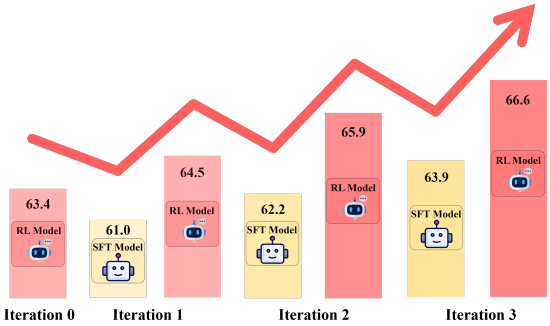


Figure 1: Iterative improvements in the average performance of SFT and RL models, reflecting progressive enhancement through self-evolution.

To address the challenges of scarce SFT data and the limited performance of the existing RL approach, we propose **EvolveSearch**, a novel iterative self-evolution framework that combines SFT and RL to enhance web search capabilities **without** any

external human-annotated reasoning data. Specifically, EvolveSearch proceeds in alternating phases: (i) In the RL exploration phase, the model interacts with a web search environment, leveraging tool-use capabilities and receiving a hybrid reward signal. This enables the model to identify and learn from high-reward rollouts. (ii) In the SFT optimization phase, the best-performing rollouts from the RL phase are selected based on three criteria and used to optimize the model via SFT, yielding a stronger initialization (*i.e.*, cold-start policy) for the next RL cycle. By iteratively alternating between exploration and optimization, EvolveSearch progressively bootstraps the performance of the RL model and the SFT model, learning robust and effective search behaviors from its own experience without human intervention, as illustrated in Figure 1.

To validate the effectiveness of EvolveSearch, we conduct extensive experiments in realistic web search settings. Evaluation on seven multi-hop question-answering (MHQA) benchmarks demonstrates EvolveSearch consistently outperforms competitive baselines. Notably, it achieves a 4.7% average accuracy gain over the previous state-of-the-art (SOTA), demonstrating the benefits of combining supervised fine-tuning and reinforcement learning in a self-evolution framework. These results highlight the strength of iterative learning from high-reward rollouts, enabling substantial performance improvements without reliance on human-annotated reasoning data.

Overall, our contributions are as follows:

- We propose EvolveSearch, a novel framework that, to the best of our knowledge, is the first to iteratively combine RL with SFT to enhance LLMs' capabilities in the web search scenario.
- EvolveSearch requires no human-annotated reasoning data; instead, it leverages high-quality rollouts from RL models to enable continuous self-improvement via self-generated supervision.
- We conduct extensive empirical evaluations on multiple MHQA datasets, demonstrating the significant effectiveness and generality of EvolveSearch over existing SOTA.

## 2 Background

In this work, a question-answering rollout expands through a ReAct-based (Yao et al., 2022) sequence of `thought-action-observation` iterations. Within each iteration, the LLM agent generates: (i) A free-form `thought` ($\tau$) to extract information, adjust action plans, and track task progress, etc. (ii) A structured `action` ($\alpha$) to interact with external environments. (iii) This interaction yields an `observation` ($o$) which serves as feedback for the next iteration. Formally, we represent the agentic execution loop at a given time step $t$ as a triplet $(\tau_t, \alpha_t, o_t)$. In EvolveSearch, the action $\alpha$ can be either *search*, corresponding to the utilization of a search tool, or *answer*, which involves formulating a response to the given question. The `observation` $o$ after a *search* action typically includes a list of relevant results, such as the top-10 titles and snippets retrieved from the search tool. Consequently, the historical rollout leading up to time step $t$ denoted as, $\mathcal{H}_t$, can be represented as the sequence:

$$\mathcal{H}_t = (\tau_0, \alpha_0, o_0, \tau_1, ..., \tau_{t-1}, \alpha_{t-1}, o_{t-1}). \quad (1)$$

At time step $t$, the agent considers the historical rollout $\mathcal{H}_t$ to generate `thought` $\tau_t$ and subsequently select an `action` $\alpha_t$, following policy $\pi(\tau_t, \alpha_t|\mathcal{H}_t)$. Then it gets a feedback `observation` $o$ if $\alpha_t$ is a *search* action. Otherwise, the rollout comes to an end after the *answer* action $\alpha_t$ is completed.

## 3 Method

**EvolveSearch** employs iterative Reinforcement Learning (RL) and Rejection Sampling Fine-tuning (RSFT) to train an LLM agent towards solving question-answering problems with multiturn thinking and tool use. We provide an overview of our framework, with the full workflow illustrated in Figure 2. We begin by evenly dividing the dataset into $N$ parts. The process then involves iteratively executing the following two stages $N$ times:

- **Stage 1**. In the i-th iteration, we utilize the i-th portion of the raw data to perform RL training on the SFT model from the previous iteration. By implementing a hybrid reward mechanism, we obtain an RL model with better generalization capabilities and rollouts during training. These rollouts provide diverse and high-quality training data for the subsequent SFT training phase.

- **Stage 2**. We merge the high-quality and diverse rollouts from the previous stage into the data pool, then refine the cold-start data by applying three filtering rules. This refined data is subsequently used for SFT of the fixed base model, thereby enabling a more robust

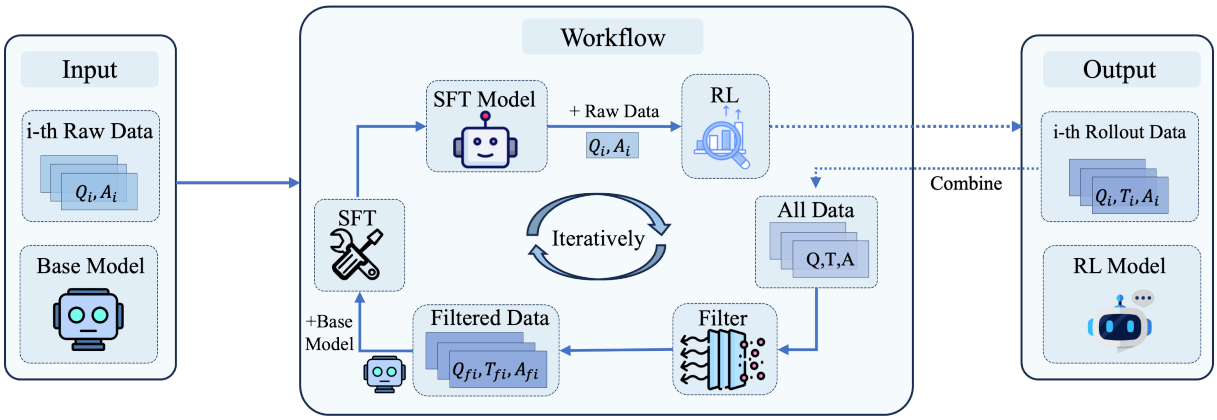


Figure 2: The overall framework of **EvolveSearch**. We iteratively input raw question-answering data and a fixed base model. In the $i$-th iteration, the base model first performs SFT with a filter pool of training data, followed by RL with the $i$-th raw data. The rollout data during RL is filtered and incorporated into the training data pool for SFT in the subsequent iteration.

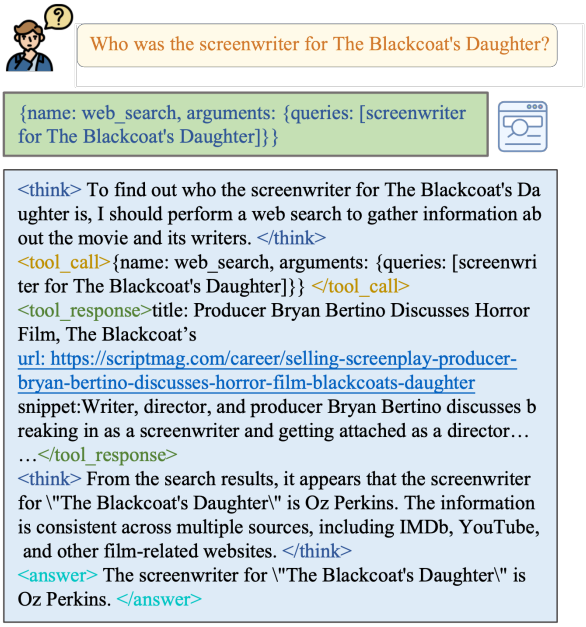


Figure 3: The illustration of a rollout that exactly matches the format.

cold-start model for RL training in the next iteration.

We will further explain these two stages in Section 3.1 and Section 3.2, respectively. The iterative algorithm workflow is presented in Appendix D.

## 3.1 Hybrid Reward Reinforcement Learning

To encourage the model to explore diverse, high-quality rollouts, we propose hybrid reward reinforcement learning: (i) A composite reward function is designed to guide the model towards proper tool calls and the derivation of accurate answers (Section 3.1.1); (ii) A modified version of Group Relative Policy Optimization (GRPO) (Shao et al., 2024) is applied to improve optimization stability and bypass the need for an extra value model (Section 3.1.2). The prompt for RL rollout is shown in Appendix A.

### 3.1.1 Reward Design

We structure reward function to comprise two key components: format reward and answer reward.

**Format Reward.** We establish a specific template to generate rollouts, as illustrated in Figure 3. According to this template, the model's `thought` for each turn is enclosed within <thinking></thinking> tags, any tool call (`action`) is placed within <tool_call></tool_call> tags, the corresponding tool feedback (`observation`) is enclosed by <tool_response></tool_response> tags, and finally, the answer is presented within <answer></answer> tags. Only when a rollout exactly matches the format, it receives a 1.0 format reward, denoted as $R_{\text{f}} = 1.0$. Otherwise, we do not further check the answer, and the rollout receives a final reward of 0.0.

**Answer Reward.** When the format is strictly followed, we employ a judge model to assess the correctness of the answer (between <answer> and </answer> tags). The judge prompt is shown in Appendix B. A rollout receives a 1.0 reward ($R_a = 1.0$) if the answer is correct, otherwise $R_a = 0.0$. In experiments, we also apply the F1 and recall as the answer reward for analysis, which we detail in Appendix F.

We define the final reward as a combination of

the above two rewards:

$$R = \begin{cases} 0.5 * (R_{\text{f}} + R_{\text{a}}), & \text{if the format is correct} \\ 0, & \text{if the format is incorrect} \end{cases} \tag{2}$$

#### 3.1.2 Group Relative Policy Optimization

In this work, we adopt the Group Relative Policy Optimization (GRPO) algorithm. GRPO optimizes the current policy $\pi_\theta$ by leveraging a reference policy $\pi_{\theta_{\text{ref}}}$ along with a set of rollouts generated by an existing policy $\pi_{\theta_{\text{old}}}$. As suggested by (Yu et al., 2025) and (Liu et al., 2025), we modify the original sample-level loss of GRPO into the token-level loss for better training performance. Specifically, given $G$ rollouts $\{y_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot|x)$(with each input $x \sim D$, where $D$ is the experience distribution), the current policy is then optimized by maximizing the following objective function:

$$\mathcal{J}(\theta) = \mathbb{E}_{x\sim D, \{y_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot|x)} \frac{1}{\sum_{i=1}^{G} |y_i|} \sum_{i=1}^{G} \sum_{t=1}^{|y_i|} \left[\min\left(r_{i,t}A_{i,t}, \text{clip}(r_{i,t}, 1-\epsilon, 1+\epsilon)A_{i,t}\right) - \beta \mathbb{D}_{\text{KL}}\right] \tag{3}$$

where $\epsilon$ is the clipping threshold and $|y_i|$ is the length of rollout $y_i$. The $\mathbb{D}_{\text{KL}}$ represents the discrepancy of the predicted probability between the current policy $\pi_\theta$ and the reference policy $\pi_{ref}$.

The advantage $A_{i,t}$ and $r_{i,t}$ are defined as follows:

$$\begin{aligned} r_{i,t} &= \frac{\pi_\theta(y_{i,t}|x, y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t}|x, y_{i,<t})} \\ A_{i,t} &= \frac{R_i - \text{mean}(\{R\})}{\text{std}(\{R\})} \end{aligned} \tag{4}$$

where $R_i$ represents reward for $y_i$, and $\text{mean}(\cdot)$, $\text{std}(\cdot)$ are calculated over the batch to normalize reward scores into advantage estimates.

### 3.2 Rejection Sampling Fine-Tuning

To enhance the utilization of rollouts during the RL phase and provide a better cold-start model for the next RL iteration, we collect the rollouts in RL and use rejection sampling fine-tuning to learn high-quality and diverse samples. The following three rules are applied sequentially to ensure high-quality, diverse and multi-step rollout filtering.

**Rule 1: High-Reward Selection (HRS).** We only retain rollouts with rewards $\geq \delta$ to ensure the high quality of training samples.

**Rule 2: Same Query Deduplication (SQD).** For multiple rollouts with the same query, we retain the sample that utilizes the tools the most, to ensure the diversity of the training samples.

**Rule 3: Multi-Calls Selection (MCS).** To enhance data utilization efficiency, we combine rollouts from the current and previous iterations. We observe that samples with multiple tool calls offer meaningful thinking and search features. As noted in Section 5, data quantity is prioritized over quality. Hence, we select the top k rollouts with the most tool calls for SFT in each iteration. The distribution of tool calls for RL rollouts in each iteration is presented in the Appendix C.

**Supervised Fine-Tuning.** After obtaining the data $D_f$ filtered by the three rules, we train the base model in an SFT manner. This produces a good cold-start model for the next RL stage. Given the question $x$ and the agentic execution rollout $\mathcal{H} = (y_0, y_1, ..., y_{n-1}, y_n)$, where each $y_i \in \{\tau, \alpha, o\}$, the loss function for SFT is computed as follows:

$$\begin{aligned} L = &- \frac{1}{\sum_{i=1}^{|\mathcal{H}|} \mathbb{I}[y_i \neq o]} \\ &\times \sum_{i=1}^{|\mathcal{H}|} \mathbb{I}[y_i \neq o] \cdot \log \pi_\theta(y_i \mid x, y_{<i}) \end{aligned} \tag{5}$$

Here, $\mathbb{I}[y_i \neq o]$ filters out tokens corresponding to external feedback, ensuring that the loss is calculated only on the actions of the agent.

## 4 Experiments

### 4.1 Benchmark and Evaluation Metrics

In EvolveSearch, we utilize the same training and testing data as DeepResearcher (Zheng et al., 2025). Specifically, for the training dataset, we used a distribution ratio of NQ (Kwiatkowski et al., 2019):TQ (Joshi et al., 2017):HotpotQA (Yang et al., 2018):2Wiki (Ho et al., 2020) as 1:1:3:3 with a total of 80,000 samples. This includes 75% of the samples from multi-hop scenarios, which better reflect the complex information-seeking behaviors required for deep research questions. For the evaluation dataset, we use the NQ, TQ, HotPot, and 2Wiki datasets as the in-domain evaluation set, totaling 2,048 examples. We use the Musique (Trivedi et al., 2022), Bamboogle (Press et al., 2022), and PopQA (Mallen et al., 2022)

datasets as the out-of-domain evaluation set, totaling 1,129 examples. We utilize a judge model to evaluate the correctness of the model's response.

### 4.2 Implementation Details

We utilize Qwen2.5-7B-Instruct[1] (Yang et al., 2024) as our backbone. During the RL training phase, each sample undergoes 16 rollouts with a training batch size of 128, a learning rate of 1e-6, a maximum search count of 10, and a temperature of 1.0. The training epoch is set to 1. We utilize Qwen2.5-72B-Instruct[2] as our judge model. We split the training data into $N = 10$ parts, and 8,000 samples are consumed for RL training in each iteration. In the data filter process, we set k to 2000 to select samples with the highest number of tool calls, $\delta$ to 0.7 to select samples with a reward exceeding 0.7. In the RSFT training phase, we employ Zero-3 offload(Aminabadi et al., 2022), with a batch size of 64, a learning rate of 3e-6, and the training epoch set to 1.

### 4.3 Baselines

To evaluate the effectiveness of our approach, we compare it with the following baseline methods:

- CoT: This baseline generates answers using Chain-of-Thought reasoning without depending on any external reference context.
- RAG: This method integrates CoT reasoning with retrieved reference context to assist in the generation.
- Search-o1 (Li et al., 2025) + Web Search: A multi-step reasoning baseline where the model is permitted to generate search queries and send real-time search requests via APIs, accessing URLs to browse web pages. The model can then generate answers based on the content of these web pages.
- Search-r1 (Jin et al., 2025): An RL-based fine-tuning strategy. During both the training and inference stages, it utilizes a retriever to access information from Wikipedia. We consider two setups: using Qwen2.5-7B-base[3] or Qwen2.5-7B-Instruct as the initial actor models, respectively.
- R1-Searcher (Song et al., 2025): Unlike Search-r1, when given a search query, it searches Bing and answers questions by summarizing the first three pages of the search results.
- DeepResearcher (Zheng et al., 2025): Unlike R1-Searcher, DeepResearcher does not restrict its search to a specific domain and allows for autonomous selection of URLs rather than mandatorily summarizing the top three search results.
- DeepResearcher + Model-Based Reward (MBR): The standard DeepResearcher uses the F1 score as a reward. For fair comparison with our method, we also use Qwen2.5-72B-Instruct as the judge model for the response reward. For simplicity, we denote this baseline as DeepResearcher*.
- RLSearch: An RL-only baseline of our framework. It is only trained via RL with the same raw data and the same hyperparameters.

All the baselines use Qwen2.5-7B-Instruct as the backbone unless specifically mentioned.

### 4.4 Main Results

The results of EvolveSearch for 3 iterations and other baselines are presented in Table 1.

**EvolveSearch consistently outperforms all baselines within training domains.** EvolveSearch achieves the highest performance across all datasets within the four domains, significantly surpassing all baselines on the NQ and 2Wiki datasets. While DeepResearcher + MBR demonstrates comparable performance on the NQ and HotpotQA datasets, it is noteworthy that DeepResearcher + MBR utilizes model-based reward, which results in performance significantly higher than when using F1 as a reward during training. Therefore, using F1 as a reward tends to shorten the model's response, thereby affecting the overall quality of its responses.

**EvolveSearch demonstrates impressive generalization capabilities in out-of-domain scenarios.** It consistently surpasses all baseline methods across three out-of-domain datasets. This indicates that EvolveSearch allows the model to effectively acquire reasoning skills that can be applied broadly, instead of just adjusting to specific training data.

[1] https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
[2] https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
[3] https://huggingface.co/Qwen/Qwen2.5-7B

| Method | Inference Environment | In Domain | | | | | Out of Domain | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | NQ | TQ | Hotpot | 2Wiki | Avg | Musique | Bamb | PopQA | Avg |
| ***Prompt Based*** | | | | | | | | | | |
| CoT† | Local RAG | 32.0 | 48.2 | 27.9 | 27.3 | 33.9 | 7.4 | 21.6 | 15.0 | 14.7 |
| CoT+RAG† | Local RAG | 59.6 | 75.8 | 43.8 | 24.8 | 51.0 | 10.0 | 27.2 | 48.8 | 28.7 |
| Search-o1† | Web Search | 55.1 | 69.5 | 42.4 | 37.7 | 51.2 | 19.7 | 53.6 | 43.4 | 38.9 |
| ***Training Based*** | | | | | | | | | | |
| Search-r1-base† | Local RAG | 60.0 | 76.2 | 63.0 | 47.9 | 61.8 | 27.5 | 57.6 | 47.0 | 44.0 |
| Search-r1-instruct† | Local RAG | 49.6 | 49.2 | 52.5 | 48.8 | 50.0 | 28.3 | 47.2 | 44.5 | 49.5 |
| R1-Searcher† | Web Search | 52.3 | 79.1 | 53.1 | 65.8 | 62.6 | 25.6 | 65.6 | 43.4 | 44.9 |
| DeepResearcher† | Web Search | 61.9 | 85.0 | 64.3 | 66.6 | 69.5 | 29.3 | 72.8 | 52.7 | 51.6 |
| DeepResearcher* | Web Search | 66.4 | 86.0 | 65.4 | 75.0 | 73.2 | 29.0 | 71.7 | 50.2 | 50.3 |
| RLSearch-ite1 | Web Search | 68.5 | 86.3 | 66.7 | 76.4 | 74.5 | 30.4 | 74.2 | 50.4 | 51.6 |
| RLSearch-ite2 | Web Search | 69.3 | 87.7 | 66.8 | 75.5 | 74.9 | 33.5 | 73.6 | 51.1 | 52.7 |
| RLSearch-ite3 | Web Search | 69.8 | 88.4 | 65.0 | 71.8 | 73.8 | 30.8 | 77.0 | 51.8 | 53.2 |
| ***Ours*** | | | | | | | | | | |
| EvolveSearch-ite1 | Web Search | 68.5 | 87.4 | 65.4 | 75.6 | 74.2 | 29.3 | 74.0 | 51.2 | 51.5 |
| EvolveSearch-ite2 | Web Search | 69.4 | 86.3 | 66.3 | **78.5** | 75.1 | 31.6 | 76.5 | **52.8** | 53.6 |
| EvolveSearch-ite3 | Web Search | **71.0** | **89.5** | **67.7** | 76.4 | **76.2** | **33.8** | **77.1** | 50.3 | **53.7** |

Table 1: Main results on seven multi-hop question answering (MHQA) benchmarks. All the results labelled with † are taken from Zheng et al. (2025).

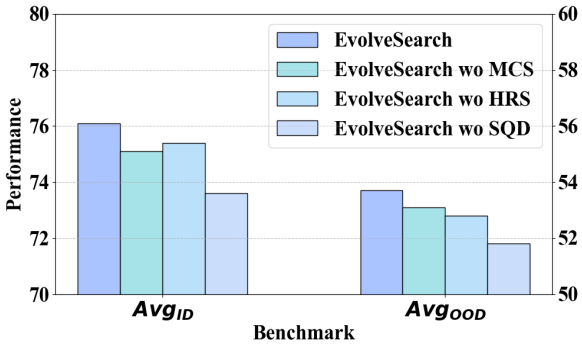


Figure 4: The impact of different data filtering rules on performance.

| Training Reward | Method | $AVG_{ID}$ | $AVG_{OOD}$ |
|---|---|---|---|
| Recall | DeepResearcher | 65.5 | 52.8 |
| | EvolveSearch-ite1 | 68.8 | 55.8 |
| | EvolveSearch-ite2 | 69.6 | 56.6 |
| | EvolveSearch-ite3 | **69.9** | **58.8** |
| F1 | DeepResearcher | 61.2 | 50.8 |
| | EvolveSearch-ite1 | 61.6 | 51.4 |
| | EvolveSearch-ite2 | **62.2** | 51.0 |
| | EvolveSearch-ite3 | 62.1 | **52.1** |
| Judge Model | DeepResearcher | 73.2 | 50.3 |
| | EvolveSearch-ite1 | 74.2 | 51.5 |
| | EvolveSearch-ite2 | 75.1 | 53.6 |
| | EvolveSearch-ite3 | **76.2** | **53.7** |

Table 2: Performance comparison of different training rewards during the training phase.

**A good cold-start model is crucial.** We select high-quality and diverse reasoning rollouts during the RL phase to employ Rejection Sampling Fine-Tuning (RSFT) for obtaining a better initial policy model. From the results, especially the comparison between the three iterations of EvolveSearch and the RLSearch baseline, we can conclude that a good cold-start model can further enhance the model's potential and stability in RL training, thereby consistently improving the performance.

## 5 Analysis

**Data filtering plays a vital role.** To demonstrate the significance of data filtering rules, we report the model's average in-domain and out-of-domain performance across different filtering rules. For fair comparison, we randomly select only 2000 SFT samples for each experiment, followed by using the additional 8000 samples for RL training. As shown in Figure 4, each data filtering rule is essential. Specifically, filtering out multi-call data aims to enhance the model's initial multi-step reasoning capabilities, filtering data with high rewards ensures data accuracy, and filtering data with different queries increases diversity, thereby comprehensively improving the model's performance.

**EvolveSearch remains effective with different training rewards.** To evaluate the effectiveness

| Method | Judge Model | $AVG_{ID}$ | $AVG_{OOD}$ |
|---|---|---|---|
| DeepResearcher* | DeepSeek-V3 | 66.1 | 38.0 |
| | chatgpt-4o-latest | 71.0 | 42.5 |
| | grok-3 | 74.5 | 45.9 |
| EvolveSearch-ite1 | DeepSeek-V3 | 66.3 | 39.8 |
| | chatgpt-4o-latest | 71.8 | 43.9 |
| | grok-3 | 74.7 | 48.0 |
| EvolveSearch-ite2 | DeepSeek-V3 | 67.4 | 41.1 |
| | chatgpt-4o-latest | 72.6 | 45.0 |
| | grok-3 | 75.3 | 48.7 |
| EvolveSearch-ite3 | DeepSeek-V3 | 67.4 | 41.6 |
| | chatgpt-4o-latest | 72.7 | 45.3 |
| | grok-3 | 75.8 | 49.3 |

Table 3: Comparison of model performance using different judge models.

of our method when employing different answer rewards, we replace the answer reward component with three common and widely recognized metrics: Recall, F1 Score, and Model-Based Reward. We utilize these metrics to compare the performance of our method against the baseline. To ensure consistency between training and testing, the same answer evaluation metric is used for both. The experimental results, presented in Table 2, demonstrate that our method consistently outperforms the baseline when using each of these different answer reward metrics on seven benchmarks. Furthermore, we observe that as the number of iterations of our method increases, the model's performance on both in-domain (ID) and out-of-domain (OOD) datasets gradually improves.

**EvolveSearch still demonstrates superior performance across different judge models.** To further verify our approach's effectiveness, we utilize different judge models to evaluate the model's response. In Table 3, we select three well-known LLMs, DeepSeek-V3 (DeepSeek-AI, 2024), chatgpt-4o-latest[4], and grok-3[5] as the judge model. Although their performance is slightly lower than the trained judge model, the improvement trend relative to the baseline remains consistent. After the first iteration of training, the model outperforms the baseline in both in-domain and out-of-domain benchmarks. As the iteration increases, the model's performance gradually improves, further demonstrating the effectiveness of the method.

[4]https://openai.com
[5]https://x.ai

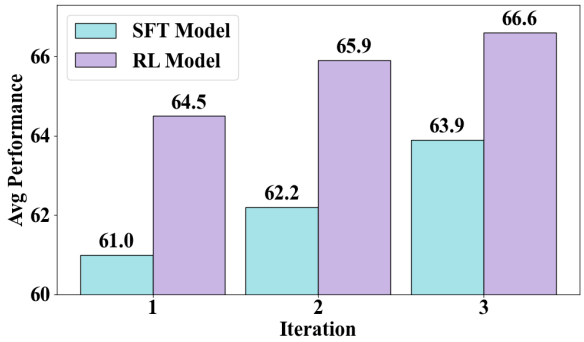


Figure 5: Performance of SFT model and RL model at different iterations.

| Training Num | $AVG_{ID}$ | $AVG_{OOD}$ |
|---|---|---|
| 4000 | 75.6 | 53.3 |
| 8000 | 75.2 | 52.6 |
| 12000 | 75.4 | 53.0 |
| 16000 | 75.4 | 53.4 |

Table 4: The impact of different data volumes on model performance during the RSFT phase.

**The performance of the SFT Model and RL Model improves as the number of iterations increases.** Figure 5 presents the performance of the SFT model and RL model across different iterations. We observe that as iterations grow, not only does the RL Model exhibit significant improvements across seven different benchmarks, but the SFT Model also shows considerable enhancement. This confirms the high quality of our chosen data and demonstrates the effectiveness of EvolveSearch.

**Iterative training increases the frequency of model tool calls.** To investigate the model's tool usage on the test set throughout the entire iterative training process, we record the average number of search tool calls by the SFT model and the RL model on the test set. As shown in Figure 6, our findings reveal that, as the number of iterations increases, the model increasingly depends on the tool, leading to gathering more information. Furthermore, we note that for the majority of questions, the model requires only three calls to the search tool to reach a solution, indicating that the training data is not sufficiently challenging. A test case of the interaction between the model and the environment is presented in the Appendix E.

**Data quality is more important than data quantity.** To investigate the impact of SFT training

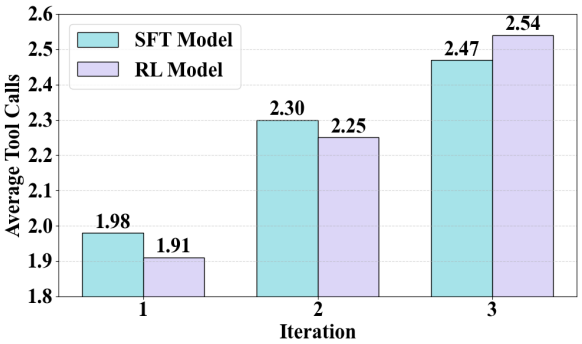


Figure 6: Average number of tool calls in the test set across different iterations.

data volume on model performance, we conduct controlled experiments using progressively scaled datasets. Based on historical RL training rollouts, we remove the filtering on the number of web search tool calls. SFT is performed at four different scales: 4,000, 8,000, 12,000, and 16,000 samples. All experiments subsequently undergo an identical RL training phase using a fixed 8,000 question-answering samples. The results are presented in Table 4, which show that merely increasing data volume does not necessarily enhance performance, which indicates that the quality of data is more important than the quantity.

## 6 Related Work

**Search Agent.** Current methods often rely on manually designed workflows to guide large language models (LLMs) in interacting with external knowledge sources (Wang et al., 2024a; Wu et al., 2025b). Recent studies like OpenResearcher (Zheng et al., 2024), IterDRAG (Yue et al., 2025), AirRAG (Feng et al., 2025), and others have improved search capabilities using these detailed workflows. However, these approaches are limited by their dependence on human-crafted prompts and interaction patterns. Recent developments about SFT for Retrieval-Augmented Generation (RAG) have become a preferred method over manual optimization (Yu et al., 2024). For example, CoRAG (Wang et al., 2024b) utilizes Monte Carlo Tree Search (MCTS) to select optimal document blocks under budget constraints but faces high computational costs and limited generalization due to reliance on supervised signals. Reinforcement Learning (RL) presents an end-to-end approach to enhance large language models' capabilities, improving reasoning skills significantly by late 2024 (Ouyang et al., 2022; Shao et al., 2024). Recent research explores RL for external knowledge retrieval, with systems like Search-R1 (Jin et al., 2025), ReSearch (Chen et al., 2025), and R1-Searcher (Song et al., 2025) evolving beyond predefined cues to models that autonomously develop reasoning during retrieval. However, these methods often converge quickly, resulting in low data efficiency and limited performance gains.

**Self-Evolution.** Large language models (LLMs) have shown the capability to annotate datasets without relying on human-annotated labels, enabling low-resource training for other LLMs. In typical setups, a larger model, the teacher, generates labels for a smaller model, the student, in a process known as *context distillation*. Various algorithms can be employed, such as conventional supervised fine-tuning (SFT) (Alpaca, 2023; Hsieh et al., 2023), in-context learning (Krishna et al., 2024), and preference optimization (Tunstall et al., 2023; Llama-3, 2024). Self-evolution methods remove the necessity for a larger LLM, reducing computational demand and API costs. Recent research has shown this approach is viable using an unlabeled dataset with a few examples for; context (Huang et al., 2022; Tian et al., 2023). For instance, (He et al., 2019) uses a small labeled dataset for initial fine-tuning before applying the trained generator to annotate the unlabeled data, similar strategies are employed for rationalization tasks in (Jie et al., 2024). (Meng et al., 2022) enhances labeled datasets with additional samples, though this is limited to classification. Our approach focuses on the open web search domain, combining SFT and RL to enhance search capabilities without requiring any external human-annotated reasoning data.

## 7 Conclusion

In an era where current search agents have surpassed the capabilities of most humans, we propose EvolveSearch, a novel iterative self-evolution framework that synergistically combines RL with SFT to enhance web search capabilities without any external human-annotated reasoning rollouts. Extensive experiments on multiple MHQA benchmarks demonstrate that EvolveSearch consistently improves performance with each iteration, ultimately achieving an average accuracy improvement of 4.7% over SOTA methods on seven benchmarks, paving the way to self-evolution and self-improvement in open web search domains.

## Limitations

EvolveSearch relies on iterative collection and filtering of the rollouts during RL training for SFT, which adds to the computation cost of the whole training process. Designing a streaming rollout filtering system with higher sample efficiency is one possible way to minimize the impact of extra computation.

In this work, the tool call is limited to web search, so the performance of other tools remains unknown. The investigation of multiple tools is left for future research.

## References


Alpaca. 2023. Introducing alpaca: A strong and performant instruction-following language model. Accessed: 2024-06-10.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. 2022. Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale. *Preprint*, arXiv:2207.00032.

Anthropic. 2025. Building effective agents.

Mingyang Chen, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. 2025. Research: Learning to reason with search for llms via reinforcement learning. *Preprint*, arXiv:2503.19470.

DeepSeek-AI. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Wenfeng Feng, Chuzhan Hao, Yuewei Zhang, Jingyi Song, and Hao Wang. 2025. Airrag: Activating intrinsic reasoning for retrieval augmented generation via tree-based search. *arXiv preprint arXiv:2501.10053*.

Junxian He, Jiatao Gu, Jiajun Shen, and Marc'Aurelio Ranzato. 2019. Revisiting self-training for neural sequence generation. *arXiv preprint arXiv:1909.13788*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2022. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*.

Yeo Wei Jie, Ranjan Satapathy, and Erik Cambria. 2024. Plausible extractive rationalization through semi-supervised entailment signal. *arXiv preprint arXiv:2402.08479*.

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

Mandar Joshi, Eunsol Choi, Daniel 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 (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Satyapriya Krishna, Jiaqi Ma, Dylan Slack, Asma Ghandeharioun, Sameer Singh, and Himabindu Lakkaraju. 2024. Post hoc explanations of language models can improve language models. *Advances in Neural Information Processing Systems*, 36.

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.

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

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*.

Llama-3. 2024. Meta llama 3. Accessed: 2024-06-10.

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

Yu Meng, Jiaxin Huang, Yu Zhang, and Jiawei Han. 2022. Generating training data with language models: Towards zero-shot language understanding. *Advances in Neural Information Processing Systems*, 35:462–477.

OpenAI. 2025. Deep research system card.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*.

Melissa Z Pan, Mert Cemri, Lakshya A Agrawal, Shuyi Yang, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Kannan Ramchandran, Dan Klein, and 1 others. 2025. Why do multiagent systems fail? In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Ji-Rong Wen, Yang Lu, and Xu Miu. 2025. R1-searcher: Incentivizing the search capability in llms via reinforcement learning.

Katherine Tian, Eric Mitchell, Huaxiu Yao, Christopher D Manning, and Chelsea Finn. 2023. Fine-tuning language models for factuality. *arXiv preprint arXiv:2311.08401*.

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.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, and 1 others. 2023. Zephyr: Direct distillation of lm alignment. *arXiv preprint arXiv:2310.16944*.

Liang Wang, Haonan Chen, Nan Yang, Xiaolong Huang, Zhicheng Dou, and Furu Wei. 2025. Chain-of-retrieval augmented generation. *arXiv preprint arXiv:2501.14342*.

Xiaohua Wang, Zhenghua Wang, Xuan Gao, Feiran Zhang, Yixin Wu, Zhibo Xu, Tianyuan Shi, Zhengyuan Wang, Shizheng Li, Qi Qian, Ruicheng Yin, Changze Lv, Xiaoqing Zheng, and Xuanjing Huang. 2024a. Searching for best practices in retrieval-augmented generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17716–17736, Miami, Florida, USA. Association for Computational Linguistics.

Ziting Wang, Haitao Yuan, Wei Dong, Gao Cong, and Feifei Li. 2024b. Corag: A cost-constrained retrieval optimization system for retrieval-augmented generation. *arXiv preprint arXiv:2411.00744*.

Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. 2024. Measuring short-form factuality in large language models. *arXiv preprint arXiv:2411.04368*.

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

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhenglin Wang, Zhengwei Tao, Ding-Chu Zhang, Zekun Xi, Xiangru Tang, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025a. Webdancer: Towards autonomous information seeking agency.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and Fei Huang. 2025b. Webwalker: Benchmarking llms in web traversal. *Preprint*, arXiv:2501.07572.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv: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 Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

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

Zhenrui Yue, Honglei Zhuang, Aijun Bai, Kai Hui, Rolf Jagerman, Hansi Zeng, Zhen Qin, Dong Wang, Xuanhui Wang, and Michael Bendersky. 2025. Inference scaling for long-context retrieval augmented generation. In *The Thirteenth International Conference on Learning Representations*.

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

Yuxiang Zheng, Shichao Sun, Lin Qiu, Dongyu Ru, Cheng Jiayang, Xuefeng Li, Jifan Lin, Binjie Wang, Yun Luo, Renjie Pan, and 1 others. 2024. Openresearcher: Unleashing ai for accelerated scientific research. *arXiv preprint arXiv:2408.06941*.

Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, and 1 others. 2023. Agents: An open-source framework for autonomous language agents. *arXiv preprint arXiv:2309.07870*.

## A RL Prompt

An example of rollout in the RL phase is shown below.

**Prompts for RL Rollout**

A conversation between User and Assistant. The user asks a question, and the assistant solves it by calling one or more of the following tools.
<tools>
{
"name": "web_search",
"description": "Utilize the web search engine to retrieve relevant information based on multiple queries.",
"parameters": {
"type": "object",
"properties": {
"queries": {
"type": "array",
"items": {
"type": "string",
"description": "The search query."
},
"description": "The list of search queries."
}
},
"required": ["queries"]
}
}
</tools>

The assistant starts with one or more cycles of ( thinking about which tool to use –> performing tool call –> waiting for tool response), and ends with ( thinking about the answer –> answer of the question). The thinking processes, tool calls, tool responses, and answer are enclosed within their tags. There could be multiple thinking processes, tool calls, tool call parameters and tool response parameters.

Example response:
<think> thinking process here </think>
<tool_call>
{"name": "tool name here", "arguments": {" parameter name here": parameter value here, " another parameter name here": another parameter value here, ...}}
</tool_call>
<tool_response>
{"name": "tool name here", "content": {"result name here": result value here, "another result name here": another result value here, ...}}
</tool_response>
<think> thinking process here </think>
<tool_call>
{"name": "another tool name here", "arguments": {...}}
</tool_call>
<tool_response>
{"name": "another tool name here", "content": {...}}</tool_response>
(more thinking processes, tool calls and tool responses here)
<think> thinking process here </think>
<answer> answer here </answer>

User: **{INPUT QUERY}** "

## B Judgement Prompt

The prompt for answer judgement in our work is based on Wei et al. (2024). The detailed prompt is shown below.

**Prompts for Answer Judgement**

Please evaluate whether the model's response is correct based on the given question, standard answer, and the model's predicted answer. Your task is to rate the result as: **Correct** or **Incorrect** .

**Correct Response**

Here are examples of Correct responses:

Question: What are Barack Obama's children's names?
Standard Answer: Malia Obama and Sasha Obama

Model Prediction 1: Malia Obama and Sasha Obama
Model Prediction 2: Malia and Sasha
Model Prediction 3: Most people would say Malia and Sasha, but I'm not sure and need to confirm.
Model Prediction 4: Barack Obama has two daughters, Malia Ann and Natasha Marian, but they are commonly known as Malia Obama and Sasha Obama.
Model Prediction 5: Barack Obama's children

These responses are Correct because:
They fully include the important information from the standard answer.
They do not contain any information that contradicts the standard answer.
Only the semantic content is considered; language ( English or Chinese), case, punctuation, grammar, and order are not important.
The presence of vague statements or guesses is acceptable, as long as the standard answer is included and there is no incorrect or contradictory information.

**Incorrect Response**

Here are examples of Incorrect responses:

Question: What are Barack Obama's children's names?
Standard Answer: Malia Obama and Sasha Obama

Model Prediction 1: Malia
Model Prediction 2: Malia, Sasha, Susan, and Sasha Obama or Malia Obama, or Natasha Marian, or Einstein
Model Prediction 3: Although I don't know their exact names, I can say that Barack Obama has two children.
Model Prediction 4: You might be thinking of Bessie and Olivia. But you should check the latest

references for detailed information. Is that the correct answer?
Model Prediction 5: Barack Obama's children

These responses are Incorrect because:
They contain factual statements that contradict the standard answer.
The answer is empty, restates the question.
The answer lists multiple answers, restates the answer.

**Special Notes**

Please note the following:
The standard answer may contain multiple aspects of the question's response, and within the same aspect, there may be multiple different descriptions, all of which are correct and are given within the same parentheses, connected by commas. For example, consider the question "What is the name of the social media platforms purchased by Elon Musk?":
Predicted answers "Twitter," "Twitter, X," and "X" are all Correct.
For standard answers that contain responses to multiple aspects of the question, the model must provide answers to all aspects to be considered correct; otherwise, it is directly judged as Incorrect. There is no such output as **Partially Correct**. These answers will be given in different parentheses. For example, consider the question " Who are the original members of the band The Beatles?":
Predicted answers "John Lennon, Paul McCartney, George Harrison, Ringo Starr" that include all answers are considered Correct.
Predicted answers like "John Lennon, Paul McCartney" that do not include all answers are considered Incorrect.

**Additional Guidelines**
Also, pay special attention to the following:
For questions with numerical standard answers, the predicted answer should match the standard answer. For example, consider the question "What is the total length of the Jinshan Railway Huangpujiang Special Bridge in meters?":
Predicted answers "3518," "3518.1," and "3518.17" are all Correct.
Predicted answers "3520" and "3600" are all Incorrect.
If the model's prediction does not directly answer the question and attempts to bypass or fails to directly provide the standard answer, it is considered an Incorrect answer.
If the standard answer contains more information than the question, the predicted answer only needs to include the information mentioned in the question.
If it is obvious from the question that the predicted answer has omitted information, it is considered Correct.
If it is clear that different translation versions of a name refer to the same person, they are also considered Correct.
You should focus more on the match between the standard answer and the model's prediction, rather than whether the standard answer is correct.

**Example Question**
Here is a new example question. Please rate the predicted answer as one of the following:
Question: {question}
Standard Answer: {target}
Predicted Answer: {predicted answer}
Only return the option represented by Correct or Incorrect, that is, only return A or B, without adding any other text.

## C Distribution of RL Rollouts

The distribution of RL Rollouts is shown in Figure 7.

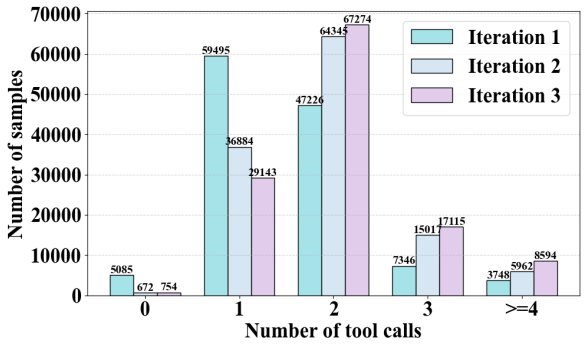


Figure 7: The distribution of tool calls for RL rollouts. As the iteration increases, the average number of tool calls also increases, and the number of trivial rollouts ($\leq$ 1 tool call) significantly decreases. This indicates that the rollouts gain in both diversity and quality during training.

**Algorithm 1** The iterative workflow of **EvolveSearch**.

**Input**: The number of iterations $N$, raw data $\mathcal{RD} = \{\{d_1, a_1\}, ..., \{d_n, a_n\}\}$, a base model $\mathcal{M}$.
**Output**: RL model $\mathbf{M}_{rN}$ in the last iteration.

$\mathcal{RD}^\star \leftarrow \{\mathcal{RD}_1^\star, ..., \mathcal{RD}_N^\star\}$ ▷ Divide the raw data evenly into $N$ parts
$\mathcal{DP} \leftarrow \{\}$ ▷ Initialize data pool
**for** $i \leftarrow 1...N$ **do**
  $\mathcal{D} \leftarrow GET(\mathcal{DP})$ ▷ Get all data
  $\mathcal{FD} \leftarrow \mathcal{F}(D)$ ▷ Filter data by three rules
  $\mathcal{M}_s \leftarrow SFT(\mathcal{M}, \mathcal{FD})$ ▷ Get SFT model
  $\mathcal{M}_{ri} \leftarrow RL(\mathcal{M}_s, \mathcal{RD}_i^\star)$ ▷ Get RL model
  $\mathcal{D}_i^r \leftarrow \{\{d_{i1}, t_{i1}, a_{i1}\}, ..., \{d_{ik}, t_{ik}, a_{ik}\}\}$ ▷ Obtain $i$-th RL rollout data
  $\mathcal{DP} \leftarrow \mathcal{DP} \cup \mathcal{D}_i^r$ ▷ Combine $\mathcal{D}$ with $\mathcal{D}_i^r$
**end for**
**return** $\mathcal{M}_{rN}$

## D The Iterative Algorithm Workflow

The overall workflow framework of **EvolveSearch** is shown in Algorithm 1.

## E Case Study

A test case of **EvolveSearch** is presented below. In this case, the model demonstrates abilities such as critical thinking and refinement through multiple tool calls, proving the effectiveness of our training method.

**EvolveSearch Case**

Query: What was the name changed to after Allison J71 was modified to a thrust of 7400 lbf?
<think>
To find the name change after Allison J71 was modified to a thrust of 7400 lbf, I need to first determine which model was created with this modification.
</think>
<tool_call>
{\"name\": \"web_search\", \"arguments\": {\"queries\": [\"Allison J71 modified to 7400 lbf thrust model name\"]}}
</tool_call>
<tool_response>
...
</tool_response>
<think>
The web search detailed various F−89 Scorpion versions but didn't mention the Allison J71 modified to 7400 lbf thrust. I'll refine the search to find the specific model name linked to this engine modification.
</think>
<tool_call>
{\"name\": \"web_search\", \"arguments\": {\"queries\": [\"Allison J71 modified to 7400 lbf thrust model name F−89 Scorpion\"]}}
</tool_call>
<tool_response>
...
</tool_response>
<think>
The web search results don't specify the model name change for the Allison J71 modified to 7400 lbf thrust but indicate it's developed from the J35 engine, particularly the J35−A−23 variant. Since the F−89 Scorpion used the J71, it's likely that the thrust modification was part of an engine upgrade rather than a model name change. I will try to identify if any specific F−89 Scorpion model included this engine modification.
</think>
<tool_call>
{\"name\": \"web_search\", \"arguments\": {\"queries\": [\"F−89 Scorpion model with Allison J71 7400 lbf thrust\"]}}
</tool_call>
<tool_response>
...
</tool_response>
<think>
The web search suggests that the Allison J35 engine with 7,400 lbf wet thrust was used in the F−89 Scorpion. Specifically, the F−89C−25−NO block adopted the J35−A−33 engine, providing a dry thrust of 5,600 lbf and 7,400 lbf with afterburner. This implies the sought name change might be tied to this engine upgrade in the F−89C series, possibly altering the model designation to F−89C−25−NO.
</think>
<answer>
The name change after modifying the Allison J71 to a thrust of 7400 lbf likely relates to the F−89C−25−NO production of the F−89 Scorpion, which used the Allison J35−A−33 engine with the needed thrust specifications.
</answer>

## F Other Answer Reward

For F1, we directly define $R_a$ as the exact F1 score of the predicted answer and the gold answer, i.e., $R_a = F1(\text{pred}, \text{gold}) \in [0, 1]$.

For recall, through some preliminary experiments, we found that a hard label is more helpful in RL rather than a soft score. Therefore, we define $R_a$ as follows:

$$R_a = \begin{cases} 1.0, & \text{if the recall score = 1.0} \\ 0.0, & \text{if the recall score < 1.0} \end{cases}$$