LLM Agents for Search via Reinforcement Learning with Trajectory-Level Self-Evaluation

Leandro Yamachita da Costa[1], João Baptista de Oliveira e Souza Filho[1]

[1]Programa de Engenharia Elétrica
Universidade Federal do Rio de Janeiro (UFRJ) – Rio de Janeiro, RJ – Brazil

leandro.yamachita@coppe.ufrj.br, jbfilho@poli.ufrj.br

Abstract. *Agentic search enables language models to iteratively query and reason over retrieved information to answer complex questions. While reinforcement learning is used to train such agents, most approaches rely solely on final-answer accuracy as the reward signal, which can limit learning. We analyze a reward strategy combining final-answer correctness with trajectory-level feedback, guiding the agent to improve its search behavior. The agent is trained with QLoRA for efficient fine-tuning and performs self-evaluation by comparing answers to references and scoring search trajectories using predefined rubrics, without external supervision. Experiments on multi-hop QA tasks show gains of up to 5.7 points over standard reward setup.*

1. Introduction

Recent advancements in the reasoning ability of LLMs have made them capable of handling increasingly complex tasks [Chen et al., 2024; El-Kishky et al., 2025]. Models such as DeepSeek-R1 [Guo et al., 2025] and OpenAI o1 [Jaech et al., 2024], classified as reasoning models, generate step-by-step chains-of-thought before providing a final answer, resulting in significant improvements across a range of benchmarks.

At the same time, there is an increasing interest in adding external tools to LLMs, such as search tools [Zheng et al., 2025], code interpreters [Wang et al., 2024], and APIs [Yao et al., 2024]. These tools expand what LLMs can do by giving them access to external and real-time information, the ability to run code, and the capacity to interact with other systems. When combined with reasoning, tool use lets LLMs act as goal-driven agents in tasks that require making a series of decisions.

Among the various tools that can augment LLMs, search tools are particularly important. While LLMs store a large amount of knowledge from pretraining, they often fall short when handling questions that require up-to-date information, specialized topics, or private information [Kandpal et al., 2023]. In these situations, outdated or missing knowledge can lead to inaccurate or fabricated answers. Integrating a search tool helps to overcome these limitations, improving the model's reliability and adaptability, especially for smaller models with more restricted internal knowledge.

A widely used method for integrating external information is Retrieval-Augmented Generation (RAG) [Lewis et al., 2020], wherein documents obtained from some retrieval system are added directly into the prompt as supporting context. However, early RAG approaches often depend on manually designed and static workflows [Yan et al., 2024, Ma et al., 2023], such as fixed query reformulation prompts or hard-coded error-checking steps, which limit their ability to generalize across tasks.

A more flexible alternative to early RAG implementations is the agentic search approach, in which the LLM acts as an interactive agent. Rather than relying on static query prompts or hard-coded workflows, the agent dynamically issues multiple search queries, reflects on the retrieved results at each step, and decides how to proceed until a final answer is reached [Jin et al., 2025, Chen et al., 2025, Song et al., 2025]. Well-known examples of this strategy are the OpenAI's Deep Research [OpenAI, 2025] and Google DeepMind's Deep Research [Google DeepMind, 2024].

Reinforcement Learning (RL) has played a key role in enhancing the reasoning capabilities of LLMs [Guo et al., 2025]. Models such as DeepSeek-R1 and OpenAI o1 demonstrate that RL can foster skills like planning, problem decomposition, and self-correction. In this paradigm, the model learns through trial and error, guided by a reward signal. This makes RL more exploratory than Supervised Fine-Tuning (SFT), which relies on large volumes of labeled data, typically annotated by domain experts.

Many recent RL approaches for LLMs use final answer accuracy as the main reward signal [Guo et al., 2025, Chen et al., 2025], typically measured by exact match, F1-score, or judged by another LLM [Su et al., 2025], depending on the task complexity. While these rewards are applied at the trajectory level, that is, assigned once per full interaction, they are typically sparse, focusing only on the correctness of the final answer. This can limit the model's ability to learn the intermediate behaviors required in multi-step tasks. Search-based agents, for instance, must learn to iteratively improve their queries, interpret search results, and construct answers in a stepwise manner. Process reward models [Lightman et al., 2023] offer an alternative by assigning rewards to intermediate steps within the decision-making process, but they require more complex credit assignment. Instead of relying only on the final answer or adopting a process-based setup, we explore a reward signal that evaluates the quality of the entire search trajectory, the sequence of queries, search results, and reasoning steps, since different trajectories can lead to the same answer. Trajectory-level evaluation gives more detailed and useful feedback on agent behavior.

In this work, we built a search agent trained with RL, where the reward reflects both the accuracy of the final answer and the quality of the search trajectory. Our setup uses self-evaluation: the model assesses its own trajectory based on predefined rubrics and evaluates the final answer using a reference, eliminating the need for external judges. We compare this agent to a baseline trained only with final-answer rewards, testing both on multi-hop question answering tasks, which require multiple search steps, and on a single-hop benchmark. Results show that using an agentic search setup with trajectory-level rewards can improve performance by up to 5.7 percentage points.

All experiments were conducted on a cost-efficient computing setup using the Quantized Low-Rank Adaptation (QLoRA) method [Dettmers et al., 2023], which reduces memory usage by combining model quantization (using 4-bit weights in this work) with small trainable adapter layers. The results also highlight that reinforcement learning can be successfully applied using lightweight Parameter-Efficient Fine-Tuning (PEFT) techniques [Xu et al., 2023].

2. Method

This section details the search agent's configuration and the reinforcement learning approach used to train the model.

2.1 Agentic Search

The architecture of the search agent used in this work is illustrated in Figure 1. The agent is a LLM with access to an external search tool, to which it can make queries and receive relevant information in response.

Given a question as input (step 1), the agent performs an iterative reasoning and search cycle (steps 2 and 3). In each iteration, it reflects on the current state of the trajectory, including previous queries and search results, and decides whether it has enough information to answer the question or if it should continue searching. The agent's reasoning must be explicitly generated within the <think>...</think> tags. If the agent decides to perform a search, it must encapsulate the query within the <search>...</search> tags. At each step, only one search is allowed.

When the agent determines that it has gathered enough information, it provides a final response within the <answer>...</answer> tags (step 4). Each action, whether a search tool call or a response, must be preceded by an explicit justification within the <think> tags, documenting the decision-making process at each step.

This process generates a complete trajectory (rollout), consisting of a sequence of reasoning, searches, and intermediate responses. The input to the LLM at each step includes the entire history of the trajectory up to that point, including the queries made, their respective results, and previous reasoning. After each search action (indicated by </search>), the model's generation is paused, the query is executed externally, and the results are incorporated into the history to form the context for the next step.

Since the model's internal knowledge may be inaccurate or outdated, the agent's response should rely solely on the information obtained through external searches. This constraint reduces the likelihood of hallucinations and increases the reliability of the responses. This approach is particularly crucial for smaller models, which, due to their limited capacity for storing knowledge, are more prone to hallucinations.

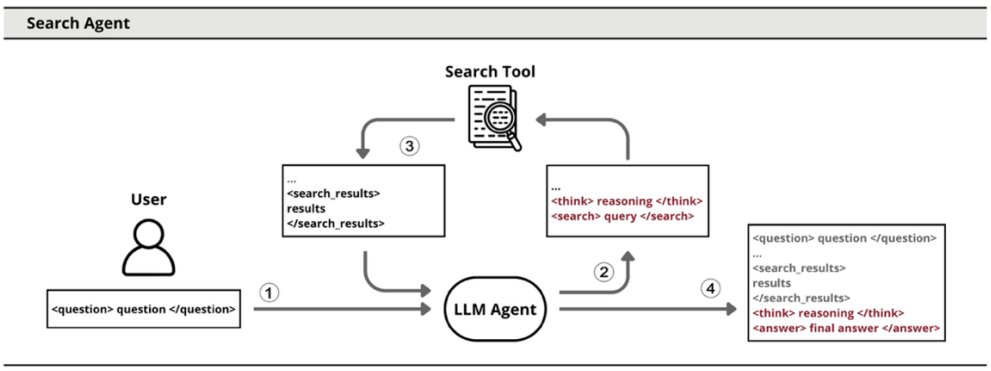

Figure 1. Agentic search pipeline.

The system prompt used by the agent is illustrated in Figure 2. It only defines general instructions and formatting rules, such as the requirement to encapsulate reasoning, searches, and responses in specific tags. No search strategy is explicitly encoded in the prompt, the agent is entirely responsible for deciding how to decompose the question, how many searches to perform, how to interpret them, and when to end the process.

System Prompt

You are a helpful assistant tasked with answering user questions accurately. You have access to an external search tool that you must use to answer the question.

Follow these strict instructions:

- Never rely on your internal knowledge; always search for external information and base your response solely on the search results.
- Always reason before taking any action, and enclose your thinking process within <think> </think> tags.
- To perform a search, enclose the search query within <search> </search> tags. You will receive search results in a <search_results> </search_results> block.
- After receiving the search results, analyze them within <think> </think> tags.
- When confident, provide a concise answer containing only the requested fact. Think inside <think> </think> tags before answering. Enclose the final answer in <answer> </answer> tags, with no explanation or extra formatting, just the answer, with no trailing spaces or newlines.

Figure 2. System prompt adopted by the search agent.

2.2 Reinforcement Learning Training

The agent was trained through Reinforcement Learning (RL), where the policy is represented by the LLM itself, and the action corresponds to the complete trajectory generated by the agent from the user's question. For this purpose, we adopted the Group Relative Policy Optimization (GRPO) algorithm [Shao et al., 2024], chosen due to its lower GPU memory requirements compared to widely used algorithms such as Proximal Policy Optimization (PPO) [Schulman et al., 2017].

Unlike PPO, which requires training an additional value function to estimate the expected return and serve as a baseline for variance reduction, GRPO derives this baseline from a group of trajectories generated per input, avoiding the need to learn a separate value network. This makes GRPO particularly well suited for low-resource settings and efficient fine-tuning of large language models.

GRPO has been effectively used in several recent studies [Ma et al., 2025, Singh et al., 2025]. The objective function used by GRPO is described as follows:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{q,\{o_i\}\sim\pi_{\theta_{\text{old}}}}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left(\min\left(\frac{\pi_\theta(o_{i,t}\mid q,o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}\mid q,o_{i,<t})}\hat{A}_{i,t},\right.\right.\right.$$
$$\left.\left.\left.\text{clip}\left(\frac{\pi_\theta(o_{i,t}\mid q,o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}\mid q,o_{i,<t})},1-\epsilon,1+\epsilon\right)\hat{A}_{i,t}\right)-\beta\,\text{KL}\left[\pi_\theta(\cdot\mid q,o_{i,<t})\,\|\,\pi_{\text{ref}}(\cdot\mid q,o_{i,<t})\right]\right)\right] \quad (1)$$

The GRPO objective optimizes a policy π_θ using updates based on group-relative advantages $\hat{A}$. Each token's update is scaled by the importance ratio between the current policy π_θ and the old policy $\pi_{\theta_{old}}$, clipped for stability. Advantages are computed by standardizing the trajectory rewards within each group of G trajectories. Each trajectory o_i represents a model-generated answer sampled for a given prompt q. The KL term penalizes divergence between the current policy and a fixed reference policy. To simplify and avoid storing a reference model, the KL term was removed in this work.

The process of calculating the advantages is illustrated in Figure 3. For each question, a total of n trajectories is generated, each receiving a scalar reward (its calculation will be detailed in the next section). Based on these rewards, advantages are computed per trajectory, reflecting its relative quality within the group.

During generation, the agent's trajectory consists of both the segments generated by the model (reasoning, queries, and final response) and the results returned by the search tool. Since these results are not produced by the model, their tokens are masked during training, meaning they are excluded from the gradient computation, ensuring that the learning process is guided solely by the decisions made by the agent.

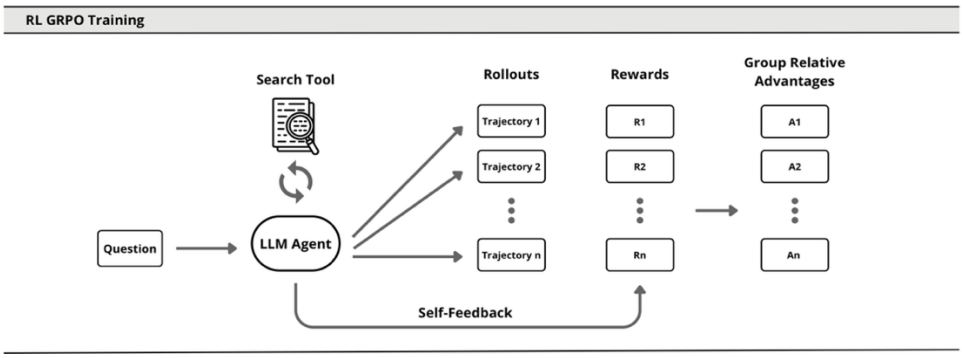

Figure 1. GRPO training pipeline

Reward Design

For the agent's training, we constructed the reward signal based on three main criteria: (i) the correctness of the final answer, which assesses whether the generated response matches the correct solution; (ii) the quality of the trajectory, measured against guidelines that identify and classify errors made throughout the trajectory; and (iii) the formatting of the response, which verifies compliance with formatting instructions. The individual contributions of each component are detailed in the following subsections, and their integration into the final reward function is described in the experimental setup.

Answer Reward

The reward associated with the final answer (R_a) is obtained by comparing the answer generated by the agent with a reference answer. We use an LLM-as-Judge approach [Gu et al., 2024], where the model under training evaluates the similarity between both answers and issues a binary verdict (True or False).

This strategy replaces rule-based metrics, such as exact match or F1-score, and enables the evaluation of longer and more complex responses with greater flexibility without the need of an additional in-memory model or external API calls. The accuracy of this self-assessment was validated by comparing the model's verdicts with those of an external evaluator based on GPT-4.1-mini. A 90% agreement between the two supports the feasibility of using the model itself as a judge during training. This comparison was performed only on the base model.

Trajectory Reward

Simply considering the correctness of the final answer is not enough to assess the agent's performance, as identical answers can be generated from substantially different trajectories. For example, a correct answer may have been obtained based on the model's internal knowledge, which is undesirable, or from a well-conducted search strategy. Similarly, an incorrect answer may result from failures in interpreting the results or from prematurely stopping the search process. Therefore, it is necessary to evaluate not only the final result but also the quality of the trajectory that led to it.

To capture these differences, we defined an additional reward (R_t) associated with the quality of the trajectory, based on the identification of errors made during the search process. The evaluation follows specific guidelines that describe the types of failures the agent may exhibit. To prevent reward hacking and unintended behaviors, we kept these

criteria simple and interpretable: (1) Critical error – Answer based on the model's internal knowledge, even when correct. Considered the most critical error, as it undermines the system's reliability; (2) Major error – It includes: (i) loss of focus on the question, i.e., the final answer does not address what was requested; and (ii) lack of effort in searches, i.e., the agent stops the process before conducting sufficient searches; (3) Moderate error – The correct information is in the results, but the agent fails to interpret it properly; (4) Minor error – Partially correct answers or responses with irrelevant information.

The trajectory score is assigned based on the identified errors: 1.0 (no errors), 0.7 (minor errors), 0.4 (moderate errors), 0.2 (major errors), and 0.0 (critical errors). These values were defined empirically based on experiments with the base model.

The score assignment, which defines the reward (R_t), is carried out by the model itself. After generating a trajectory, the agent receives its own sequence of actions along with the reference answer as input and must issue an explained score. However, a high variability was observed in the evaluations produced by the base model. To increase consistency, the model was subjected to supervised fine-tuning with 5,000 annotated examples, using evaluations generated by GPT-4.1 as references. The training was carried out with QLoRA for 3 epochs, resulting in an RMSE of 0.12 on a set of 500 samples separated for testing.

Format Reward

We also introduced a reward (R_f) to assess whether the trajectory follows the required format, which is essential for extracting queries and identifying the final answer.

At each step of the trajectory, it was checked whether the model correctly used the mandatory <think> tags (always required) and, depending on the type of action, the <search> or <answer> tags. When all tags are correctly used, the step receives a reward of 0.2; otherwise, the reward is zero. The final reward associated with the trajectory corresponds to the average of the rewards obtained at each step.

3. Experimental Design

This section will describe the experimental setup used.

Base model and Datasets

We implemented the agent using the Qwen2.5-7B-Instruct model [Qwen, 2024] and trained it on MuSiQue [Trivedi et al., 2022], a Wikipedia-based dataset requiring multi-hop reasoning across multiple passages. Each example includes a main question, sub-questions, supporting documents, and the final answer. The agent received only the main question and interacted iteratively with a search tool to construct its response. MuSiQue is well-suited to our setup, as it encourages multiple cycles of search and reasoning, typically requiring two hops, with a maximum of four. To evaluate generalization, we tested the agent on MoreHopQA [Schnitzler et al., 2024], another Wikipedia-based multi-hop benchmark that includes a subset of MuSiQue questions and requires inference beyond explicit retrieval; we ensured that overlapping examples were excluded from training. As a single-hop contrast, we used Pirá [Paschoal et al., 2021], a domain-specific dataset of questions from scientific abstracts on ocean and climate topics. Each dataset was evaluated on 1,000 samples.

Search Tool

In the experiments, we used a search tool based on the Dense Passage Retriever (DPR) [Karpukhin et al., 2020], with embeddings generated by the *e5-base-v2* model [Wang et al., 2022], which is specifically trained for passage retrieval. The search is performed via vector similarity between the query generated by the agent and the documents in the corpus, returning the top three most relevant passages per query for MuSiQue, and only the single most relevant passage for the other datasets, as they are simpler and contain considerably fewer passages.

For each dataset, we constructed a corpus using the contexts available within the datasets: approximately 100k passages for MuSiQue, 800 for MoreHopQA, and 650 for Pirá. This approach ensures that the documents used during the experiments are closely aligned with the domain of each task.

Baselines

We compare our approach against several baselines, all based directly on the Qwen2.5-7B-Instruct base model, without any intermediate SFT: (i) the base model without access to the search tool, answering directly from its internal knowledge; (ii) a traditional RAG configuration, where the question itself is used as the query and the retrieved documents are directly included in the model's input context; and (iii) the base model with iterative access to the search tool, without training; (iv) the search agent trained with RL using only final-answer accuracy as the reward signal.

Rewards

We tested two reward configurations. In the first, only the evaluation of the final answer and formatting were considered, with a reward of 1 assigned if the answer was deemed correct, as shown in Equation 2.

$$R = \begin{cases} 1 + R_f & \text{, if answer is correct} \\ R_f & \text{, if answer is incorrect} \end{cases} \quad (2)$$

We also tested incorporating trajectory quality evaluation into the reward. For this reward, the correctness signal of the final answer (R_t) was also used, ensuring that the model still prioritizes its correctness, as shown in Equation 3.

$$R = \begin{cases} 0.5 + R_t + R_f & \text{, if answer is correct} \\ R_t + R_f & \text{, if answer is incorrect} \end{cases} \quad (3)$$

In this configuration, the initial model is the base model with the LoRA adapter, trained via SFT, as described in the Trajectory Reward section, to improve its ability to evaluate the quality of the trajectories. Note that it is only in this configuration that we use the base model with the SFT trained QLoRA adapter prior RL.

Although the trajectory level reward function evaluates the quality of the entire trajectory, the approach used in this work does not follow a process-level RL setup where rewards are assigned and used to update the policy at each intermediate step. Instead,

rewards, both for the final answer and the trajectory, are computed only after the full trajectory is completed, and policy updates occur post-trajectory.

Training Configuration

We trained the agent using RL with 4k samples from the MuSiQue training set. The training was conducted over a single epoch, with batches of 32 samples and 5 rollouts per sample, totaling 125 update steps. The inputs were limited to 6144 tokens or 10 agent steps, and the outputs to 1024 tokens. During trajectory generation, temperature 0.7 was used; for judging responses and trajectories, the model operated with a temperature of 0.1. The learning rate was 1e-6, and the KL divergence term was disabled ($\beta = 0$).

We applied QLoRA with 4-bit quantization, using LoRA adapters with rank 32 and $\alpha = 32$. Training was done on an Nvidia H100 80GB GPU, accessed through the Hyperbolic AI Cloud platform (hyperbolic.xyz).

4. Results and Analysis

The experimental results, measured in terms of answer accuracy, are presented in Table 1. The agent trained with trajectory-based rewards achieved the best performance on both multi-hop datasets, with a notable gain on MuSiQue, where it outperformed the agent trained solely with final-answer rewards by up to 5.7 percentage points. As expected, the agentic setup proved more effective for tasks requiring multiple successive queries to construct the final answer. Figure 4 shows the progression of the three reward components during training with trajectory-level reward signals. We observe that the agent quickly learns the correct formatting to follow.

Table 1. Experimental results

	MuSiQue	MoreHopQA	Pirá
No search	0.129	0.115	0.265
Simple RAG	0.138	0.142	**0.588**
Agentic Base	0.232	0.253	0.455
Agentic RL (final answer reward)	0.290	0.291	0.524
Agentic RL (trajectory reward)	**0.347**	**0.335**	0.570

To better understand the performance gains from agent training, we analyzed how different types of errors, especially errors such as incorrect formatting, token limit overflows, or excessive steps, impact the results. This analysis used a test set of 1,000 samples from the MuSiQue dataset, though similar patterns appeared in the other datasets. Table 2 presents the percentage distribution of these errors across the three evaluated agents, including both search process errors and cases of incorrect final answers.

In the base agent, 12% of the errors were caused by the absence of valid actions (no search or answer in certain steps), often due to incorrect use of tags. Since these trajectories do not result in interpretable actions, they are treated as incorrect. After

training, this type of error was almost eliminated, which helps explain part of the performance improvement: attempts that were previously discarded due to formatting issues are now included in the evaluation process.

Figure 4. Reward components during training, with Exponential Moving Average (EMA) for smoothing.

On the other hand, the agent trained solely with the final-answer reward exhibited 18% of its errors due to exceeding the maximum number of tokens or allowed steps, suggesting that it enters search loops, repeatedly attempting to gather more information without success. Even when these limits were increased, the results remained nearly unchanged. This behavior may be related to the agent making incorrect assumptions and persisting with poorly targeted queries or struggling to recognize when it has already acquired sufficient information. Another contributing factor may be limitations in the retriever, as it often returns the same passages even after query reformulations, making it difficult to progress when the relevant evidence is absent from the retrieved results.

In the agent trained with trajectory-based rewards, these errors were almost eliminated, suggesting that it learned a more efficient search strategy. It's important to note, however, that the reduction of these errors explains only part of the performance gains: a well-formatted response can still be incorrect, and long trajectories may still fail, even with more generous step or context limits.

In summary, the results indicate that incorporating a more informative reward based on the evaluation of the full trajectory contributes to better agent performance in iterative search tasks. It is worth noting, however, that reward signals based solely on the final answer are less informative and may require increased exploration or additional rollouts per question to achieve comparable performance. Owing to computational constraints, these variants were not explored in this work but remain promising directions for future research.

Table 2. Search process errors across agents.

	Wrong answer	Wrong Formatting	Max tokens or steps	Total
Agentic Base	653 (85%)	92 (12%)	23 (3%)	768
Agentic RL (final answer reward)	582 (82%)	2 (0%)	126 (18%)	710
Agentic RL (trajectory reward)	649 (100%)	2(0%)	2 (0%)	653

5. Related Work

5.1 Training LLMs via Reinforcement Learning

With the emergence of models like DeepSeek-R1 and OpenAI o1, several studies have begun exploring how training with RL can enhance the reasoning capabilities of LLMs. Much of this research has focused on tasks amenable to automatic verification, such as mathematics [Hu et al., 2025], programming [El-Kishky et al., 2025], and logic [Xie et al., 2025]. Other domains are also being actively explored, including financial reasoning [Liu et al., 2025] and the integration of external tools [Qian et al., 2025, Feng et al., 2025].

Explored techniques include approaches that do not rely on external rewards - typically leveraging only the model's confidence in its responses [X. Zhao et al., 2025] - as well as self-learning strategies, such as those in [A. Zhao et al., 2025], where the model is trained to propose and solve its own tasks to continuously improve.

5.2 Training Search Agents via Reinforcement Learning

Some works have explored the use of RL to train LLMs as search agents [Zheng et al., 2025, Jin et al., 2025, Chen et al., 2025, Song et al., 2025]. In [Song et al., 2025], the training is divided into two stages: first, the agent learns how to operate the search tool; then, it learns how to leverage the retrieved information to answer questions accurately. In contrast, [Sun et al., 2025] simulates a search engine using an LLM, thereby eliminating dependence on external tools or specific databases.

In general, most of these works rely on rewards based solely on the final answer, verified through rules or external models, which makes it more challenging for the agent to learn effective decision-making throughout the search process.

6. Conclusions and Limitations

We implemented a search agent trained via RL using QLoRA. Given a question, the agent iteratively performs reasoning and search steps until it gathers sufficient information to answer. We compared two reward configurations: one based solely on final answer accuracy, and another that also incorporates the quality of search trajectory. The model performs self-evaluation using a reference answer and predefined criteria. Results indicate that including trajectory quality leads to superior performance, suggesting that richer reward signals can enhance agent effectiveness.

Due to the high computational cost, we did not perform multiple runs of each experiment to assess statistical significance. Nevertheless, preliminary reruns showed low variability in results, supporting the representativeness of the reported outcomes. Future work could explore deploying the agent in real-world environments - such as web search engines or big data platforms - moving beyond the simplified retriever and single search tool used in this study. In such scenarios, the agent can leverage reward rubrics tailored to specific tasks or domains, thereby enabling more targeted, trustworthy, and effective learning.

Acknowledgments

To CNPq, FAPERJ, and CAPES. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

References

Chen, Junying, et al. "Huatuogpt-o1, towards medical complex reasoning with llms." arXiv preprint arXiv:2412.18925 (2024).

Chen, Mingyang, et al. "Learning to reason with search for llms via reinforcement learning." arXiv preprint arXiv:2503.19470 (2025).

Dettmers, Tim, et al. "Qlora: Efficient finetuning of quantized llms." Advances in neural information processing systems 36 (2023): 10088-10115.

El-Kishky, Ahmed, et al. "Competitive programming with large reasoning models." arXiv preprint arXiv:2502.06807 (2025).

Feng, Jiazhan, et al. "Retool: Reinforcement learning for strategic tool use in llms." arXiv preprint arXiv:2504.11536 (2025).

Google DeepMind. Gemini Deep Research. Google, 11 Dec. 2024, https://gemini.google/overview/deep-research/.

Gu, Jiawei, et al. "A survey on llm-as-a-judge." arXiv preprint arXiv:2411.15594 (2024).

Guo, Daya, et al. "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning." arXiv preprint arXiv:2501.12948 (2025).

Hu, Jingcheng, et al. "Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model." arXiv preprint arXiv:2503.24290 (2025).

Jaech, Aaron, et al. "Openai o1 system card." arXiv preprint arXiv:2412.16720 (2024).

Jin, Bowen, et al. "Search-r1: Training llms to reason and leverage search engines with reinforcement learning." arXiv preprint arXiv:2503.09516 (2025).

Kandpal, Nikhil, et al. "Large language models struggle to learn long-tail knowledge." International Conference on Machine Learning. PMLR, 2023.

Karpukhin, Vladimir, et al. "Dense Passage Retrieval for Open-Domain Question Answering." EMNLP (1). 2020.

Lewis, Patrick, et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks." Advances in neural information processing systems 33 (2020): 9459-9474.

Lightman, Hunter, et al. "Let's verify step by step." The Twelfth International Conference on Learning Representations. 2023.

Liu, Zhaowei, et al. "Fin-r1: A large language model for financial reasoning through reinforcement learning." arXiv preprint arXiv:2503.16252 (2025).

Ma, Peixian, et al. "Sql-r1: Training natural language to sql reasoning model by reinforcement learning." arXiv preprint arXiv:2504.08600 (2025).

Ma, Xinbei, et al. "Query rewriting in retrieval-augmented large language models." Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. 2023.

OpenAI. Deep Research System Card. OpenAI, 25 Feb. 2025, https://cdn.openai.com/deep-research-system-card.pdf.

Paschoal, André FA, et al. "Pirá: A bilingual portuguese-english dataset for question-answering about the ocean." Proceedings of the 30th ACM International Conference on Information & Knowledge Management. 2021.

Qian, Cheng, et al. "Toolrl: Reward is all tool learning needs." arXiv preprint arXiv:2504.13958 (2025).

Schnitzler, Julian, et al. "Morehopqa: More than multi-hop reasoning." arXiv preprint arXiv:2406.13397 (2024).

Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

Shao, Zhihong, et al. "Deepseekmath: Pushing the limits of mathematical reasoning in open language models." arXiv preprint arXiv:2402.03300 (2024).

Singh, Joykirat, et al. "Agentic reasoning and tool integration for llms via reinforcement learning." arXiv preprint arXiv:2505.01441 (2025).

Song, Huatong, et al. "R1-searcher: Incentivizing the search capability in llms via reinforcement learning." arXiv preprint arXiv:2503.05592 (2025).

Su, Yi, et al. "Crossing the Reward Bridge: Expanding RL with Verifiable Rewards Across Diverse Domains." arXiv preprint arXiv:2503.23829 (2025).

Sun, Hao, et al. "Zerosearch: Incentivize the search capability of llms without searching." arXiv preprint arXiv:2505.04588 (2025).

Team, Qwen. "Qwen2.5 technical report." arXiv preprint arXiv:2412.15115 (2024).

Trivedi, Harsh, et al. "MuSiQue: Multihop Questions via Single-hop Question Composition." Transactions of the Association for Computational Linguistics 10 (2022): 539-554.

Wang, Liang, et al. "Text embeddings by weakly-supervised contrastive pre-training." arXiv preprint arXiv:2212.03533 (2022).

Wang, Xingyao, et al. "Openhands: An open platform for ai software developers as generalist agents." arXiv preprint arXiv:2407.16741 (2024).

Xie, Tian, et al. "Logic-rl: Unleashing llm reasoning with rule-based reinforcement learning." arXiv preprint arXiv:2502.14768 (2025).

Xu, Lingling, et al. "Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment." arXiv preprint arXiv:2312.12148 (2023).

Yan, Shi-Qi, et al. "Corrective retrieval augmented generation." (2024).

Yao, Shunyu, et al. "tau-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains." arXiv preprint arXiv:2406.12045 (2024).

Zhao, Andrew, et al. "Absolute zero: Reinforced self-play reasoning with zero data." arXiv preprint arXiv:2505.03335 (2025).

Zhao, Xuandong, et al. "Learning to reason without external rewards." arXiv preprint arXiv:2505.19590 (2025).

Zheng, Yuxiang, et al. "Deepresearcher: Scaling deep research via reinforcement learning in real-world environments." arXiv preprint arXiv:2504.03160 (2025).