

# LeTS: Learning to Think-and-Search via Process-and-Outcome Reward Hybridization

Qi Zhang<sup>1\*</sup>, Shouqing Yang<sup>1</sup>, Lirong Gao<sup>1</sup>, Hao Chen<sup>1</sup>,  
Xiaomeng Hu<sup>1</sup>, Jinglei Chen<sup>2</sup>, Jiexiang Wang<sup>2</sup>, Sheng Guo<sup>2</sup>,  
Bo Zheng<sup>2</sup>, Haobo Wang<sup>1†</sup>, Junbo Zhao<sup>1†</sup>

<sup>1</sup>Zhejiang University, <sup>2</sup>MYbank, Ant Group,  
{cheung\_se, wanghaobo, j.zhao}@zju.edu.cn,

## Abstract

Large language models (LLMs) have demonstrated impressive capabilities in reasoning with the emergence of reasoning models like OpenAI-o1 and DeepSeek-R1. Recent research focuses on integrating reasoning capabilities into the realm of retrieval-augmented generation (RAG) via outcome-supervised reinforcement learning (RL) approaches, while the correctness of intermediate think-and-search steps is usually neglected. To address this issue, we design a process-level reward module to mitigate the unawareness of intermediate reasoning steps in outcome-level supervision without additional annotation. Grounded on this, we propose **Learning to Think-and-Search (LeTS)**, a novel framework that hybridizes stepwise process reward and outcome-based reward to current RL methods for RAG. Extensive experiments demonstrate the generalization and inference efficiency of **LeTS** across various RAG benchmarks. In addition, these results reveal the potential of process- and outcome-level reward hybridization in boosting LLMs’ reasoning ability via RL under other scenarios. The code is released at <https://github.com/Cheungki/LeTS>.

## 1 Introduction

Large language models (LLMs) (Yang et al., 2024; Grattafiori et al., 2024; Zhao et al., 2023) have exhibited remarkable performance across a wide range of downstream tasks, such as code generation (Jiang et al., 2024), mathematical reasoning (Luo et al., 2023), and question answering (Yue, 2025). However, relying solely on parametric knowledge stored within LLMs presents inherent limitations, such as generating hallucinated or outdated information (Huang et al., 2025; Du et al., 2024). To mitigate these issues, integrating external knowledge sources through *retrieval-augmented*

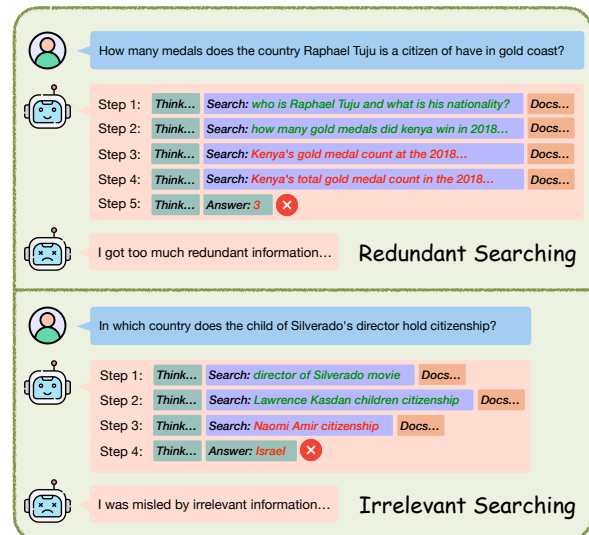


Figure 1: Illustrations of redundant searching issue and irrelevant searching issue in the current learning to think-and-search methods.

generation (RAG) has become a widely adopted and effective paradigm (Shuster et al., 2021; Gao et al., 2023). In particular, RAG empowers LLMs to dynamically access relevant information, thereby enhancing factual accuracy and adaptability in open-domain applications.

Despite its success, vanilla RAG methods struggle when faced with complicated or ambiguous queries (Lewis et al., 2020; Guu et al., 2020), where a single-shot retrieval is often insufficient for multi-hop reasoning. To address this challenge, prompting-based methods have been proposed to equip LLMs with query decomposition capabilities by manually designing prompts (Chan et al., 2024; Liu et al., 2024; Ma et al., 2023). However, such approaches heavily depend on the inherent capacity of the language model and often fail to generalize across diverse scenarios. Alternatively, training-based approaches have attempted to distill this ability from larger models into smaller ones via supervised fine-tuning (Asai et al., 2023; Yao

\*Work done during internship at MYbank, Ant Group

†Corresponding Author

et al., 2023), yet they still suffer from limited generalization and lack robust reasoning supervision.

Building upon the success of large reasoning models such as OpenAI-o1 (Jaech et al., 2024) and DeepSeek-R1 (Guo et al., 2025), reinforcement learning (RL) techniques—particularly PPO and GRPO (Shao et al., 2024)—have shown strong potential in enhancing the reasoning capabilities of LLMs. Motivated by this, recent studies have applied RL to develop the *retrieving-when-needed* behavior in LLMs, often framed as a *think-and-search* process (Song et al., 2025; Jin et al., 2025; Chen et al., 2025). While these outcome-based methods have achieved promising results, they still struggle with two common issues that lead to suboptimal performance: *i*) **Redundant Searching**, where the model repeatedly retrieves similar or unnecessary information across steps; and *ii*) **Irrelevant Searching**, where irrelevant content is introduced during intermediate steps, ultimately misleading the model. These issues are empirically illustrated in Figure 1, and highlight the limitations of using coarse-grained, outcome-level rewards alone to supervise multi-step reasoning processes.

To better supervise the intermediate think-and-search steps, we introduce two complementary rule-based process-level reward modules to reform outcome-supervised GRPO approaches. First, a knowledge redundancy reward penalizes steps that retrieve information already covered by superior rollouts. Second, a knowledge match reward identifies and rewards correct actions in weaker rollouts by comparing them against high-performing rollouts. Grounded on this, we further propose **LeTS**, a novel RL framework that hybridizes process- and outcome-level reward via an advantage rescaling approach. Notably, as illustrated in Figure 2, **LeTS** resolves the dilemma of knowledge redundancy and demonstrates strong performance on both base and instruct models. In general, through extensive experiments on numerous RAG benchmarks, **LeTS** shows its effectiveness, strong generalization, as well as inference efficiency under various scenarios.

To sum up, our contributions can be concluded in threefold:

- We propose a novel RL framework **LeTS** for RAG, which first integrates rule-based process reward with outcome-level reward under the scenarios of RAG with reasoning.
- **LeTS** effectively elicits the think-and-search

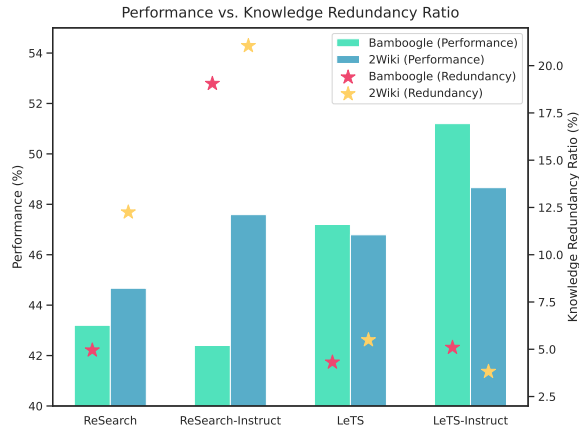


Figure 2: Collaborative comparison of performance and knowledge redundant frequency on two representative datasets with Qwen-7B-Base and Qwen-7B-Instruct as the backbone model.

ability of LLMs and achieves **2.61%** performance gain across various RAG benchmarks. Moreover, the average number of generated tokens and search times is reduced by **11.15%** and **30.85%**, respectively.

- The strong performance and inference efficiency of **LeTS** highlight the potential of process- and outcome-level reward hybridization in boosting LLMs’ reasoning ability via RL under other scenarios.

## 2 Related Work

### 2.1 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) improves the factual accuracy of language models by incorporating external knowledge into the generation process. Early methods such as REALM (Gua et al., 2020) and RAG (Lewis et al., 2020) adopt a fixed retrieve-then-generate framework, which struggles with multi-hop reasoning due to static retrieval. To address this limitation, iterative approaches like IR-CoT (Trivedi et al., 2022a) and Iter-RetGen (Shao et al., 2023) retrieve evidence step-by-step using chain-of-thought prompting. While effective, these methods often depend on prompt engineering and large black-box LLMs. Recent work introduces learning-based retrieval policies to improve flexibility. ReSearch (Chen et al., 2025), for instance, uses RL to determine when to retrieve. However, it relies solely on outcome-level rewards, ignoring the quality of intermediate reasoning steps.

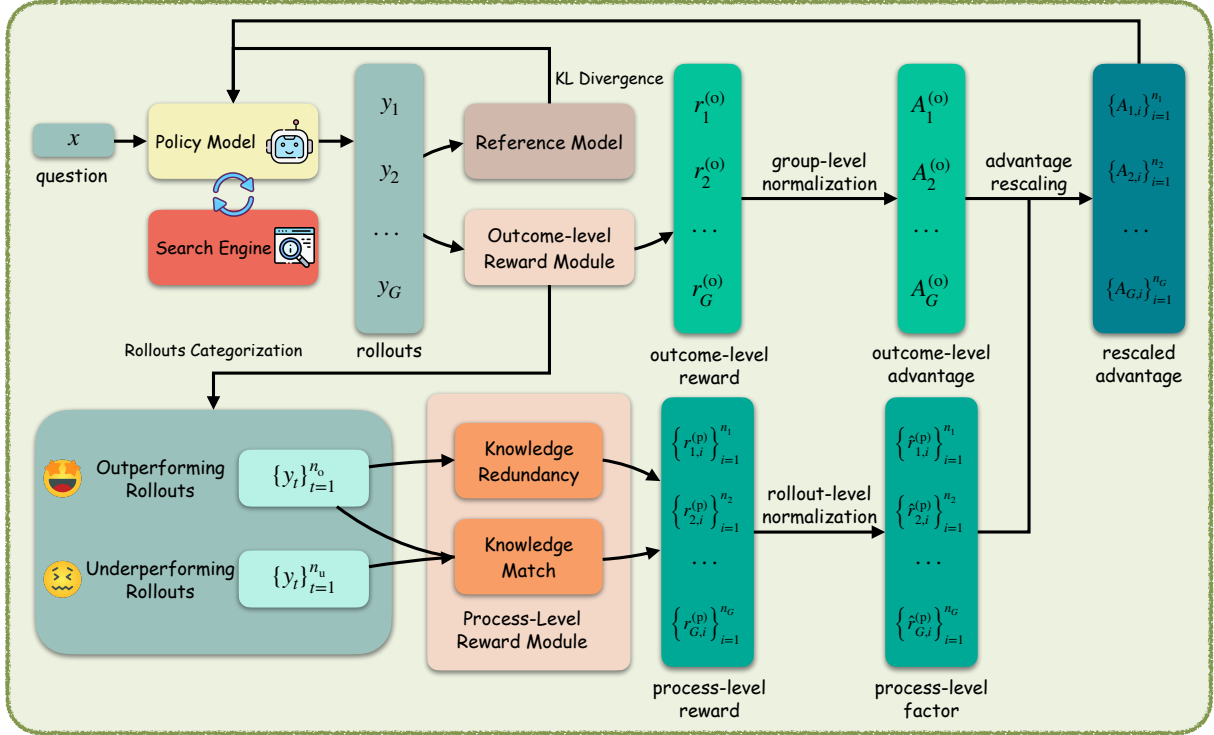


Figure 3: The core framework of **LeTS** extends conventional outcome-supervised GRPO by introducing a fine-grained categorization of rollouts within each group. Specifically, generated rollouts are classified as either outperforming or underperforming, based on their relative returns. Separate reward functions are then applied to each category to compute process-level rewards. Finally, the rollout-level advantages are rescaled at the step level using an advantage normalization procedure, enabling more precise credit assignment during policy updates.

## 2.2 Large Language Models and Reinforcement Learning

Reinforcement learning (RL) has been widely applied to teach LLMs retrieval-augmented reasoning. Prior methods such as ReSearch (Chen et al., 2025), R1-Searcher (Song et al., 2025), and SearchR1 (Jin et al., 2025) rely on outcome-level rewards to guide retrieval behavior. However, they apply coarse-grained signals uniformly across reasoning steps, making it hard to distinguish informative from redundant actions. To address this, S<sup>2</sup>R (Ma et al., 2025) introduces process-level supervision by training models to self-verify and self-correct using both process- and outcome-level rewards. While effective for structured reasoning tasks like math, its rewards are limited to verification quality and do not explicitly supervise retrieval behavior.

In contrast, our method designs rule-based, retrieval-specific process-level rewards, evaluating each think-and-search step for relevance, redundancy, and coherence. Combined with outcome-level feedback, this enables fine-grained control and more robust learning.

## 3 Methodology

In this section, we formalize the think-and-search task setting and introduce our proposed method, **LeTS**, in detail. Specifically, **LeTS** is primarily grounded in Group Relative Policy Optimization (GRPO), which estimates value baselines using a group of rollouts, in contrast to Proximal Policy Optimization (PPO) that relies on a separately trained critic model. The overall framework is illustrated in Figure 3, and a detailed pseudocode implementation is provided in Appendix D.

### 3.1 Think-and-Search Formulation

Under our think-and-search scenarios, the language model  $\mathcal{M}$  invokes the retriever  $\mathcal{R}$  selectively, only when additional external information is deemed necessary. At each step, the retrieved documents are concatenated with the original context and used as input for the subsequent reasoning stage. For a given question  $q$ , the generation process with  $n$  think-and-search steps is defined as:

$$(t_i, s_i) = \mathcal{M}\left(q, \{(t_j, s_j, d_j)\}_{j=1}^{i-1}\right) \quad (1)$$

$$d_i = \mathcal{R}(s_i)$$

, where  $t_i$ ,  $s_i$ , and  $d_i$  denote the thinking process, searching query, and retrieved documents of the  $i$ -th step, respectively. After completing  $n$  such steps, the model produces the final reasoning trace  $t_{n+1}$  and then generate the final answer  $a$  as follow:

$$(t_{n+1}, a) = \mathcal{M}\left(q, \{(t_j, s_j, d_j)\}_{j=1}^n\right) \quad (2)$$

A detailed instantiation of the think-and-search process is provided in Appendix C.1.

### 3.2 Outcome-Level Reward Modeling

Building on prior work in outcome-supervised RL (Chen et al., 2025; Jin et al., 2025; Song et al., 2025), we define an outcome-level reward based on both the structural correctness of the reasoning format and the accuracy of the final answer. For format correctness, we verify whether the generated rollout adheres to the think-and-search reasoning chain described in Section 3.1, as well as whether the final answer is properly structured. For answer correctness, we use the F1 score between the predicted answer and the gold reference, as exact match metrics are known to induce reward hacking behaviors (Song et al., 2025). The outcome-level reward  $r^o$  is then formulated as follow:

$$r^o = \begin{cases} 0.9 \cdot \text{F1}(a_{\text{pred}}, a_{\text{gt}}) + 0.1, & \text{correct format} \\ 0, & \text{wrong format} \end{cases} \quad (3)$$

, where  $\text{F1}(a_{\text{pred}}, a_{\text{gt}})$  denotes the F1 score between the predicted answer  $a_{\text{pred}}$  and the golden answer  $a_{\text{gt}}$ .

### 3.3 Process-Level Reward Modeling

Notably, relying solely on outcome-level rewards can lead to overlooking the correctness of intermediate reasoning steps (e.g., redundant operations in an otherwise successful rollout, or correct substeps within a failed one). To address this limitation, we introduce a process-level reward modeling framework tailored to our think-and-search RAG setting.

Based on the outcome-level reward defined in Equation 3, we categorize rollouts generated by the policy model during online RL into three types: (i) invalid rollouts that fail to conform to the required reasoning or answer format, (ii) outperforming rollouts that produce exact-match answers with correct format, and (iii) underperforming rollouts, which include all remaining cases that do not meet the above criteria.

Owing to the difficulty of reliably segmenting reasoning steps in malformed outputs, we omit the

process-level reward for rollouts with the wrong format. For the latter two kinds of rollouts, we design two types of rule-based process-level reward: *rollout-level knowledge redundancy reward* and *group-level knowledge match reward* to address the two phenomena mentioned in Figure 1.

#### Rollout-Level Knowledge Redundancy Reward

It’s well-established that redundant knowledge in the RAG system can increase inference latency and even ruin the generation quality (Bian et al., 2024; Chen et al., 2023; Tan et al., 2024). To mitigate this issue, we introduce a rollout-level knowledge redundancy reward that penalizes reasoning steps that retrieve documents redundant with respect to earlier steps within the same rollout. To isolate the effect of the policy model from potential retriever bias, we compute pairwise redundancy only between the current and preceding steps. Specifically, we use the Jaccard score to quantify redundancy between two sets of retrieved documents. The process-level reward for the  $j$ -th step of the  $i$ -th rollout is thus defined as:

$$r_{i,j}^p = 1 - \max_{t=1}^{j-1} \left\{ \frac{|d_{i,j} \cap d_{i,t}|}{|d_{i,j} \cup d_{i,t}|} \right\} \quad (4)$$

where  $d_{i,j}$  represents the documents retrieved in the  $j$ -th step of the  $i$ -th rollouts. In this way, steps that introduce new external knowledge will be rewarded while the redundant ones will be punished.

#### Group-Level Knowledge Match Reward

In contrast to outperforming rollouts, underperforming rollouts often suffer from irrelevant or inconsistent document retrieval (Yoran et al., 2024). Inspired by self-consistency approaches used during inference (Wang et al., 2023), we propose a group-level knowledge match (KM) reward that leverages outperforming rollouts to supervise underperforming ones through a self-contrastive mechanism within RL. Given an outperforming rollout  $y_o$  and an underperforming rollout  $y_u$ , we compute the document-level Jaccard similarity between each pair of steps  $(s_{u,j}, s_{o,k})$  from  $y_u$  and  $y_o$ , respectively:

$$\text{Match}(s_{u,j}, s_{o,k}) = \frac{|d_{u,j} \cap d_{o,k}|}{|d_{u,j} \cup d_{o,k}|} \quad (5)$$

This yields a match matrix  $\mathbf{m}^{u,o} \in \mathbb{R}^{n_u \times n_o}$ , where  $n_u$  and  $n_o$  denote the number of think-and-search steps in  $y_u$  and  $y_o$ , respectively. To derive the optimal alignment between steps, we adopt the Kuhn-Munkres algorithm (also known as the Hungarian

algorithm), which computes the maximum-weight bipartite matching between the two sets of steps. This naturally aligns with the framework of optimal transport (Cuturi, 2013), where steps in  $y_u$  are optimally mapped to those in  $y_o$  under a transport cost defined by their pairwise similarity. The resulting process-level reward vector for the underperforming rollout is given by:

$$\left\{ r_{u,j}^p \right\}_{j=1}^{n_u} = \text{Kuhn-Munkres}(\mathbf{m}^{u,o}) \quad (6)$$

, where each entry  $m_{j,k}^{u,o}$  is defined as  $\text{Match}(s_{u,j}, s_{o,k})$ . If multiple outperforming rollouts exist within the group, we select the one yielding the highest total process-level reward for alignment. If no valid outperforming rollout is available (*i.e.*, no exact-match answer), the KM reward is omitted for that group.

### 3.4 Process-and-Outcome Reward Hybridization

While outcome-supervised RL offers strong performance with low annotation cost, it suffers from a coarse reward assignment: the outcome-level reward is uniformly applied across all tokens or steps within a rollout, thereby overlooking the relative importance of individual reasoning steps. To mitigate this limitation, we propose a fine-grained integration of process-level signals into the RL objective.

Specifically, we extend the GRPO framework by hybridizing outcome- and process-level rewards through a mechanism we call **stepwise outcome-level advantage rescaling**. This approach adjusts the outcome-level advantage at each reasoning step according to the corresponding process-level reward, enabling more precise advantage assignment and better alignment between learning signals and reasoning quality.

Following the conventional outcome-supervised GRPO framework, we first compute the outcome-level advantage for each rollout by normalizing its outcome reward within the group. Specifically, for the  $i$ -th rollout in a group of size  $G$ , the outcome-level advantage is given by:  $A_i^o = \frac{r_i^o - \text{mean}(\{r_j^o\}_{j=1}^G)}{\text{std}(\{r_j^o\}_{j=1}^G)}$ . The outcome-level advantage  $A_i^o$  is then uniformly applied to all think-and-search steps within the rollout. In contrast, for the process-level reward, we adopt a local normalization strategy. Unlike process-supervised GRPO (Shao et al., 2024), which may rely on global reward signals,

our process-level supervision is designed to capture the relative importance of each step within a single reasoning chain. Thus, for the  $j$ -th step in the  $i$ -th rollout, the normalized process-level reward is computed as:

$$\hat{r}_{i,j}^p = \frac{r_{i,j}^p - \text{mean}(\{r_{i,j}^p\}_{j=1}^{n_i})}{\text{std}(\{r_{i,j}^p\}_{j=1}^{n_i})},$$

where  $n_i$  denotes the number of think-and-search steps in rollout  $i$ . This formulation allows for fine-grained reward shaping within each rollout, facilitating more targeted advantage assignment across the reasoning trajectory.

Since the normalized process-level reward  $\hat{r}^p$  captures only the relative importance of steps within a single rollout, we use it to modulate the global outcome-level advantage. Specifically, we apply a stepwise rescaling mechanism that adjusts the original advantage at each step based on its process-level signal. The rescaled advantage  $A_{i,j}$  for the  $j$ -th step in the  $i$ -th rollout is defined as:

$$A_{i,j} = \left(1 + \text{sgn}(A_i^o) \cdot \lambda \hat{r}_{i,j}^p\right) A_i^o \quad (7)$$

, where function  $\text{sgn}(A_i^o)$  denotes the sign of the outcome-level advantage and  $\lambda$  is a scaling factor controlling the magnitude of rescaling. This formulation amplifies or attenuates the original advantage based on stepwise quality, while preserving the sign of the overall trajectory’s reward signal.

For robustness, we do not apply rescaling to the final reasoning step and to rollouts excluded from process-level reward computation (*e.g.*, those with invalid formats), ensuring consistent gradient propagation in these cases.

### 3.5 Reinforcement Learning Formulation

We now formalize the overall training objective of **LeTS**. Given a policy model  $\pi_{\theta_{\text{old}}}$  and a reference model  $\pi_{\theta_{\text{ref}}}$ , based on  $G$  rollouts  $\{y_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|x)$  for input  $x \sim \mathcal{D}$ , the objective of **LeTS** is to optimize the policy  $\pi_{\theta}$  by maximizing the following objective:

$$\begin{aligned} \mathcal{J}(\theta) = & \mathbb{E}_{x \sim \mathcal{D}, \{y_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|x)} \frac{1}{G} \sum_{i=1}^G \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \\ & \left[ \min \left( \frac{\pi_{\theta}(y_{i,t}|x, y_{i,<t})}{\pi_{\text{old}}(y_{i,t}|x, y_{i,<t})} \hat{A}_{i,t}, \right. \right. \\ & \left. \left. \text{clip} \left( \frac{\pi_{\theta}(y_{i,t}|x, y_{i,<t})}{\pi_{\text{old}}(y_{i,t}|x, y_{i,<t})}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_{i,t} \right) \right. \\ & \left. - \beta \mathbb{D}_{\text{KL}}(\pi_{\theta} | \pi_{\text{ref}}) \right] \quad (8) \end{aligned}$$

Model	HotpotQA		2Wiki		MusiQue		Bamboogle		Average	
	<i>EM</i>	<i>LJ</i>	<i>EM</i>	<i>LJ</i>	<i>EM</i>	<i>LJ</i>	<i>EM</i>	<i>LJ</i>	<i>EM</i>	<i>LJ</i>
<i>Qwen2.5-7B-Base/Instruct</i>										
Direct	19.18	30.64	25.76	27.87	3.76	10.38	10.40	22.40	14.78	22.82
Naïve RAG	31.90	49.59	25.78	29.52	6.21	12.78	20.80	32.00	21.17	30.97
Iter-RetGen	34.36	52.22	27.92	31.86	8.69	16.14	21.60	35.20	23.14	33.86
IR-CoT	30.33	52.06	21.57	30.65	6.99	14.19	24.80	36.80	20.92	33.43
ReSearch-Qwen-7B	40.57	60.26	44.67	50.06	21.68	32.19	43.20	54.40	37.53	49.23
ReSearch-Qwen-7B-Instruct	<b>43.52</b>	<u>63.62</u>	<u>47.59</u>	<u>54.22</u>	<u>22.30</u>	33.43	42.40	54.40	38.95	<u>51.42</u>
<b>LeTS-Qwen-7B</b>	40.62	61.00	46.79	52.21	22.18	<b>34.09</b>	<u>47.20</u>	<u>57.60</u>	<u>39.20</u>	51.22
<b>LeTS-Qwen-7B-Instruct</b>	<u>43.16</u>	<b>63.69</b>	<b>48.66</b>	<b>55.26</b>	<b>23.21</b>	<u>33.64</u>	<b>51.20</b>	<b>62.40</b>	<b>41.56</b>	<b>53.75</b>
<i>Qwen2.5-3B-Instruct</i>										
Direct	16.00	24.56	24.81	26.47	2.28	6.04	2.40	7.20	11.37	16.07
Naïve RAG	27.97	42.61	24.38	27.46	5.75	10.05	9.60	17.60	16.93	24.43
Iter-RetGen	30.05	45.17	26.10	29.35	7.12	12.04	12.00	17.60	18.82	26.04
IR-CoT	27.48	50.67	21.66	34.14	6.74	14.78	22.40	33.60	19.57	33.30
ReSearch-Qwen-3B	30.60	-	33.79	-	14.56	-	33.60	-	28.14	-
ReSearch-Qwen-3B-Instruct	<u>35.62</u>	<u>54.40</u>	<u>39.26</u>	<u>46.10</u>	<u>17.34</u>	<b>27.10</b>	<u>37.60</u>	48.80	<u>32.45</u>	<u>44.10</u>
<b>LeTS-Qwen-3B</b>	31.02	-	36.04	-	15.76	-	34.40	-	29.31	-
<b>LeTS-Qwen-3B-Instruct</b>	<b>37.06</b>	<b>55.21</b>	<b>41.04</b>	<b>47.48</b>	<b>17.50</b>	<u>26.93</u>	<b>38.40</b>	<b>51.20</b>	<b>33.50</b>	<b>45.21</b>

Table 1: Performance of our method compared to other methods with Qwen2.5-7B-Base/Instruct and Qwen2.5-3B-Base/Instruct as the backbone model on four multi-hop benchmarks. The best results are highlighted in **bold**. Specifically, *EM* and *LJ* refer to Exact Match and LLM-as-a-Judge, respectively.

, where  $\hat{A}_{i,t} = A_{i,j(t)}$  is the rescaled advantage for the step that token  $y_{i,t}$  belongs to, and  $\epsilon$  is the clipping threshold. The mapping  $j(t)$  associates each token position  $t$  with its corresponding think-and-search step index within the rollout. The KL divergence term regularizes the policy to remain close to the reference model, weighted by the coefficient  $\beta$ . Following common practice (Jin et al., 2025; Chen et al., 2025; Song et al., 2025), we mask out retrieved tokens during training to prevent noise in the retrieval content from skewing the learning signal.

## 4 Experiments

### 4.1 Experimental Settings

**Datasets** For training **LeTS**, we largely adopt the experimental setup of ReSearch (Chen et al., 2025) for consistency and reproducibility. Specifically, we use the training split of MusiQue (Trivedi et al., 2022b), as it contains diverse types of multi-hop questions and has been constructed through rigorous quality control procedures. As for the evaluation benchmarks and metrics, please refer to A.

**Baselines** We compare **LeTS** against several representative baselines: (1) **Direct** uses the instruction-tuned model to generate answers without any retrieval. (2) **Naïve RAG** appends retrieved documents to the input query and generates answers in a single forward pass. (3) **Iter-RetGen** (Shao et al., 2023) alternates between retrieval and generation in a step-by-step manner. (4) **IR-CoT** (Trivedi et al., 2022a) interleaves retrieval with chain-of-thought prompting to support multi-hop reasoning. (5) **ReSearch** (Chen et al., 2025) applies outcome-level RL to optimize retrieval strategies. We adopt ReSearch as our primary strong baseline due to its strong empirical performance and publicly available implementation. Other comparable methods, such as **Search-R1** (Jin et al., 2025) and **R1-Searcher** (Song et al., 2025) follow similar paradigms and are therefore subsumed under ReSearch in our evaluation.

**Implementation Details** We conduct our experiments and evaluations on Qwen2.5-7B, Qwen2.5-7B-Instruct, and Qwen2.5-3B-Instruct. For the codebase, we utilize two widely adopted frame-

works: VerL\* for RL training, and FlashRAG† for evaluation. All models are trained on the MusiQue training set for 2 epochs.

For the rescaling factor  $\lambda$ , we perform hyperparameter tuning over  $\{0.05, 0.1, 0.2\}$  using Qwen2.5-7B-Instruct, and set  $\lambda = 0.1$  for all experiments in this paper. Additional implementation details of **LeTS** and the hyperparameter configurations for GRPO can be found in Appendix B.

## 4.2 Main Results

The main results of our method, compared against other baselines on Qwen2.5-7B-Base/Instruct and Qwen2.5-3B-Instruct, are presented in Table 1. Overall, our proposed **LeTS** consistently outperforms all baselines, demonstrating its effectiveness in equipping LLMs with think-and-search capabilities. Specifically, LeTS-Qwen-7B achieves average improvements of 1.67% and 1.99%, while LeTS-Qwen-7B-Instruct achieves 2.61% and 2.33% over previous RL-based methods using the same backbone, measured by *EM* and *LJ*, respectively.

Notably, **LeTS** trained on Qwen2.5-7B-Base achieves performance comparable to methods trained on Qwen2.5-7B-Instruct. In addition, our method performs well on smaller models (*i.e.*, Qwen2.5-3B-Instruct), indicating that instruct-tuned models of a smaller scale can also benefit from our proposed hybrid reward framework. Specifically, for Qwen2.5-3B-Base, we conduct 30 additional training steps with a retrieval reward only to enable its retrieval ability.

## 4.3 Generalization on Single-Hop Benchmarks

To further evaluate the generalization ability of **LeTS** in single-hop scenarios, we conduct experiments on three widely used single-hop benchmarks: NQ, PopQA, and TriviaQA. As shown in Table 2, **LeTS** continues to outperform other baselines, demonstrating its effectiveness beyond multi-hop settings.

## 5 Ablation Study and Analysis

In this section, we present ablation studies and detailed analytical experiments to further investigate the performance of **LeTS**. Unless otherwise stated, all experiments are conducted using Qwen2.5-7B-Instruct.

\*<https://github.com/volcengine/verl>

†<https://github.com/RUC-NLPIR/FlashRAG>

Method	NQ	PopQA	TriviaQA	Average
Direct	14.24	15.11	43.60	24.32
Naïve RAG	36.26	39.93	61.24	45.81
Iter-RetGen	38.59	42.78	62.90	48.09
IR-CoT	23.24	34.44	53.97	37.22
ReSearch	40.86	<b>44.58</b>	63.71	49.72
<b>LeTS</b>	<b>42.58</b>	43.91	<b>64.63</b>	<b>50.37</b>

Table 2: Performance comparison between **LeTS** with other baselines on single-hop benchmarks with Qwen-7B-Instruct as the backbone model.

## 5.1 Impact of Process-Level Reward

We conduct ablation experiments to examine the effectiveness of the two types of process-level rewards. As shown in Table 3, both the rollout-level knowledge redundancy reward and the group-level knowledge match reward significantly reduce the model’s average search time while maintaining strong overall performance.

Method	Settings		AST↓	EM↑
	w/. CR	w/. CM		
vanilla GRPO	✗	✗	3.21	22.18
w/. KR	✓	✗	2.71	22.22
w/. KM	✗	✓	2.59	22.67
<b>LeTS</b>	✓	✓	<b>2.21</b>	<b>23.21</b>

Table 3: The ablation on two types of process-level reward on the MusiQue dataset. *AST* stands for the average search time for each method across all data points. To be specific, vanilla GRPO here refers to the GRPO algorithm that solely relies on outcome-level supervision.

Moreover, Figure 4 illustrates the training dynamics of both the reward signals and knowledge redundancy. The training reward improves with the inclusion of either process-level component, and their combined usage leads to a more stable and consistent reward trajectory during RL training.

We further analyze the evolution of knowledge redundancy frequency across the four methods. Here, knowledge redundancy frequency refers to the rate at which identical documents are retrieved multiple times within a single rollout. As shown in Figure 5, models trained with the in-rollout knowledge redundancy reward (*i.e.*, w/. KR and **LeTS**) consistently exhibit lower redundancy frequencies.

Datasets		HotpotQA	2WikiMultiHopQA	MusiQue	Bamboogle	Average
Num. of Tokens	ReSearch	278.25	328.53	328.14	232.87	291.95
	LeTS	244.29	275.82	335.87	189.30	261.32
	$\Delta$ (%)	$\downarrow$ 12.20	$\downarrow$ 16.04	$\uparrow$ 2.36	$\downarrow$ 18.71	$\downarrow$ 11.15
Num. of Search Times	ReSearch	2.78	3.48	3.21	2.58	3.01
	LeTS	1.84	2.31	2.21	1.94	2.08
	$\Delta$ (%)	$\downarrow$ 33.81	$\downarrow$ 33.62	$\downarrow$ 31.15	$\downarrow$ 24.81	$\downarrow$ 30.85

Table 4: Average number of tokens and search times of **LeTS** comparing to ReSearch. Notably, tokens from the retrieved documents are not counted here, since they are not generated by the language models.

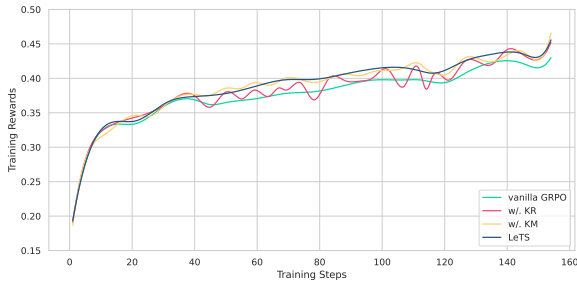


Figure 4: Training reward during training for the four methods.

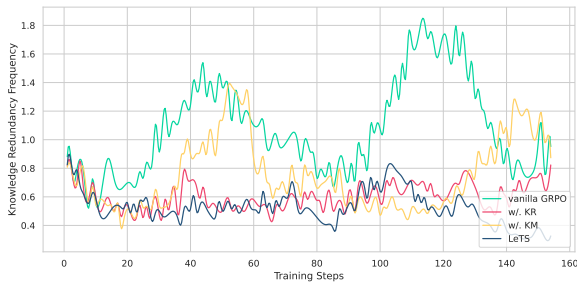


Figure 5: Knowledge redundancy frequency of the four methods during training.

## 5.2 Alleviation of the Redundant Searching Phenomenon

To assess the impact of redundant searching on the performance of retrieval-augmented language models, we analyze the average frequency of redundant document retrieval. As shown in Table 5, both baseline models tend to retrieve more redundant information when they ultimately fail to answer questions correctly. In contrast, our proposed **LeTS** significantly reduces in-rollout knowledge redundancy across four multi-hop benchmarks. These findings are consistent with the intended effect of the in-rollout knowledge redundancy reward.

Dataset		HotpotQA	2Wiki	MusiQue	Bamboogle
Correct	ReSearch	2.39	3.52	1.58	2.40
	LeTS	0.23	0.37	0.21	0.47
Incorrect	ReSearch	3.53	4.49	2.94	3.21
	LeTS	0.50	0.61	0.32	0.62

Table 5: Average in-rollout knowledge redundancy frequency of the two methods. We separately list the knowledge redundancy frequency of both correct and incorrect generations.

## 5.3 Alleviation of the Irrelevant Searching Phenomenon

To further validate the effectiveness of our approach in mitigating irrelevant searching, we evaluate the relevance of intermediate search queries generated by **LeTS** compared to other methods. Appendix E presents two case studies that qualitatively compare our method with prior approaches. As shown in Table 10, **LeTS** consistently produces more concise and relevant search queries.

## 5.4 Efficiency Analysis

We further evaluate the inference efficiency of our method on four multi-hop datasets. As shown in Table 4, **LeTS** substantially reduces both the number of generated tokens and the number of search steps during inference. In particular, **LeTS** demonstrates higher efficiency compared to prior methods by generating fewer tokens throughout the reasoning process. In terms of average search times, **LeTS** achieves a reduction of approximately 30%, without compromising performance across all four benchmarks. These results suggest that our reinforcement learning framework—built on process- and outcome-level reward hybridization—more effectively guides language models to learn efficient think-and-search behavior.



## 6 Conclusion

In this paper, we first reveal two types of drawbacks in the current outcome-supervision RL for RAG. To address these points, we design two kinds of process-level rewards. Subsequently, we propose **LeTS**, a novel RL framework that hybridizes the process- and outcome-based reward via an advantage rescaling approach. We further present the generalization and inference efficiency of our methods under different language models and numerous single-hop and multi-hop RAG benchmarks.

## Limitations

Despite **LeTS** achieving significant results on the Qwen-7B and Qwen-3B series, due to cost constraints, we have not been able to validate our approach on larger language models. Overall, our research empirically showcases the superiority of introducing the process-level supervision to outcome-based GRPO.

## Acknowledgement

This work is mainly supported by the Pioneer R&D Program of Zhejiang (No. 2024C01035). This work was supported by MYbank, Ant Group. Junbo Zhao is also supported by the NSFC under Grants (No. 62402424) and the Fundamental Research Funds for the Zhejiang Provincial Universities (226-2024-00049).

## References

- Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.
- Ning Bian, Hongyu Lin, Peilin Liu, Yaojie Lu, Chunkang Zhang, Ben He, Xianpei Han, and Le Sun. 2024. Influence of external information on large language models mirrors social cognitive patterns. *IEEE Transactions on Computational Social Systems*, pages 1–17.
- Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. 2024. Rq-rag: Learning to refine queries for retrieval augmented generation. *arXiv preprint arXiv:2404.00610*.
- Hung-Ting Chen, Fangyuan Xu, Shane A Arora, and Eunsol Choi. 2023. Understanding retrieval augmentation for long-form question answering. *arXiv preprint arXiv:2310.12150*.
- 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.
- Marco Cuturi. 2013. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.
- Xuefeng Du, Chaowei Xiao, and Sharon Li. 2024. Halo-scope: Harnessing unlabeled llm generations for hallucination detection. *Advances in Neural Information Processing Systems*, 37:102948–102972.
- Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2:1.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shitong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.
- 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.
- Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and 1 others. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55.
- Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.
- Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

- 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.
- 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, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.
- Yanming Liu, Xinyue Peng, Xuhong Zhang, Weihao Liu, Jianwei Yin, Jiannan Cao, and Tianyu Du. 2024. Ra-isf: Learning to answer and understand from retrieval augmentation via iterative self-feedback. *arXiv preprint arXiv:2403.06840*.
- Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.
- Ruotian Ma, Peisong Wang, Cheng Liu, Xingyan Liu, Jiaqi Chen, Bang Zhang, Xin Zhou, Nan Du, and Jia Li. 2025. S<sup>2</sup>r: Teaching llms to self-verify and self-correct via reinforcement learning. *arXiv preprint arXiv:2502.12853*.
- Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5303–5315.
- Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Hannaneh Hajishirzi, and Daniel Khoshdel. 2022. When not to trust language models: Investigating effectiveness and limitations of parametric and non-parametric memories. *arXiv preprint*.
- Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, Singapore. Association for Computational Linguistics.
- Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294*.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y.K. Li, and Daya Guo Y. Wu. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models.
- Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval augmentation reduces hallucination in conversation. *arXiv preprint arXiv:2104.07567*.
- 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.
- Hexiang Tan, Fei Sun, Wanli Yang, Yuanzhuo Wang, Qi Cao, and Xueqi Cheng. 2024. Blinded by generated contexts: How language models merge generated and retrieved contexts when knowledge conflicts? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6207–6227, Bangkok, Thailand. Association for Computational Linguistics.
- Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022a. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509*.
- Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022b. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 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. 2023. React: Synergizing reasoning and acting in language models, 2023. URL <https://arxiv.org/abs/2210.03629>.

Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. 2024. Making retrieval-augmented language models robust to irrelevant context. In *The Twelfth International Conference on Learning Representations*.

Murong Yue. 2025. A survey of large language model agents for question answering. *arXiv preprint arXiv:2503.19213*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and 1 others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).

## A Experimental Settings

**Evaluation Benchmarks** For evaluation on multi-hop scenarios, we consider four benchmark datasets: HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), MusiQue (Trivedi et al., 2022b), and Bamboogle (Press et al., 2023). Evaluation is conducted on the development sets of HotpotQA, 2WikiMultiHopQA, and MusiQue, and on the test set of Bamboogle, which contain 7405, 12576, 2417, and 125 samples, respectively. We also evaluate our method on three single-hop question answering datasets: NQ (Kwiatkowski et al., 2019), PopQA (Mallen et al., 2022), and TriviaQA (Joshi et al., 2017).

**Evaluation Metrics** We evaluate our method using the commonly adopted Exact Match (*EM*) metric, which considers a prediction correct only if it exactly matches the ground truth answer. To account for the overly strict nature of exact match in real-world scenarios, we additionally employ LLM-as-a-Judge (*LJ*) for automatic evaluation. In this setting, gpt-4o-mini is used to assess the semantic correctness of the final answer. The prompt used for *LJ* is provided in Appendix C.2.

## B Implementation Details

Our training is conducted on 8 Nvidia A100 GPUs, with full parameter optimization and gradient checkpointing. The detailed hyperparameter settings are listed in Table 6.

Parameter	Value
Learning Rate	1e-6
Train Batch Size	256
Number of Training Epochs	2
Number of Rollouts	5
Rollout Temperature	1.0
KL Loss Coefficient	0.001
Clip Ratio	0.2
Num. Document Retrieval	5
Rescale Factor $\lambda$	0.1

Table 6: Implementation details of LeTS.

## C Prompt Templates

In this appendix, we provide detailed prompt templates used in various stages of our experiments. These templates ensure the consistency and reproducibility of our experimental setup.

### C.1 Prompt Templates for Training

The training prompt templates are designed to clearly guide the language models in performing structured reasoning and retrieval tasks under both base and instruction-tuned settings. Specifically, Table 7 illustrates the prompt used for base models, which frames the interaction as a conversation and explicitly defines the tags for reasoning, searching, and answering. Table 8 shows the system prompt for instruction-tuned models, where the model is guided to follow a similar reasoning-and-search pattern with clear structural conventions. These prompts are essential to elicit the think-and-search behavior in a controlled and consistent manner across training scenarios.

### C.2 Prompt Templates for LLM-as-a-Judge Evaluation

The evaluation prompt in Table 9 defines the judgment criteria and expected output format for LLM-as-a-Judge, ensuring consistent and interpretable assessment of generated answers with respect to semantic correctness.

### Prompt Template For Base Model

A conversation between User and Assistant. The user asks a question, and the assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. During thinking, the assistant can invoke the wikipedia search tool to search for fact information about specific topics if needed. The reasoning process and answer are enclosed within `<think>` `</think>` and `<answer>` `</answer>` tags respectively, and the search query and result are enclosed within `<search>` `</search>` and `<result>` `</result>` tags respectively. For example, `<think>` This is the reasoning process. `</think>` `<search>` search query here `</search>` `<result>` search result here `</result>` `<think>` This is the reasoning process. `</think>` `<answer>` The final answer is `\boxed{answer here}` `</answer>`. In the last part of the answer, the final exact answer is enclosed within `\boxed{}` with latex format. User: `prompt`. Assistant:

Table 7: Prompt Template For Base Model.

## D Pseudo Code for LeTS

The pseudo code for LeTS is shown in Algorithm D.

### System Prompt Template For Instruction-Tuned Model

You are a helpful assistant that can solve the given question step by step with the help of the wikipedia search tool. Given a question, you need to first think about the reasoning process in the mind and then provide the answer. During thinking, you can invoke the wikipedia search tool to search for fact information about specific topics if needed. The reasoning process and answer are enclosed within `<think>` `</think>` and `<answer>` `</answer>` tags respectively, and the search query and result are enclosed within `<search>` `</search>` and `<result>` `</result>` tags respectively. For example, `<think>` This is the reasoning process. `</think>` `<search>` search query here `</search>` `<result>` search result here `</result>` `<think>` This is the reasoning process. `</think>` `<answer>` The final answer is `\boxed{answer here}` `</answer>`. In the last part of the answer, the final exact answer is enclosed within `\boxed{}` with latex format.

Table 8: System Prompt Template For Instruction-Tuned Model.

### Algorithm 1 Process-and-Outcome Reward Hybridized GRPO in LeTS

**Require:** Policy model  $\pi_{\theta_{\text{old}}}$ , reference model  $\pi_{\theta_{\text{ref}}}$ , dataset  $\mathcal{D}$ , rescale factor  $\lambda$ , KL penalty  $\beta$

- 1: **for** each input  $x \sim \mathcal{D}$  **do**
- 2:     Sample  $G$  rollouts  $\{y_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(x)$
- 3:     **for** each rollout  $y_i$  **do**
- 4:         Compute outcome-level reward  $r_i^{(o)}$  via Eq. 3
- 5:         **if** rollout is well-formatted **then**
- 6:             **if**  $y_i$  is in outperforming group **then**
- 7:                 Compute stepwise process rewards  $r_{i,j}^{(p)}$  using Eq. 4
- 8:             **else**
- 9:                 Compute  $r_{i,j}^{(p)}$  via group match reward using Eq. 5
- 10:             **end if**
- 11:         **end if**
- 12:         Normalize  $\{r_i^{(o)}\}_{i=1}^G$  to get  $A_i^{(o)}$
- 13:         Normalize  $\{r_{i,j}^{(p)}\}_{j=1}^{n_i}$  to get  $\hat{r}_{i,j}^{(p)}$
- 14:         **for** each step  $j$  **do**
- 15:              $A_{i,j} \leftarrow \left( 1 + \text{sgn}(A_i^{(o)}) \cdot \lambda \hat{r}_{i,j}^{(p)} \right) \cdot A_i^{(o)}$  ▷ Eq. 7
- 16:         Compute GRPO loss using  $A_{i,j}$  via Eq. 8
- 17:     **end for**
- 18: **end for**
- 19: **end for**

### Prompt for LLM-as-a-Judge

You will be given a question and its ground truth answer list, where each item can be a ground truth answer. Provided a pred\_answer, you need to judge if the pred\_answer correctly answers the question based on the ground truth answer list. You should first give your rationale for the judgment, and then give your judgment result (i.e., correct or incorrect).

Here is the criteria for the judgment: 1. The pred\_answer doesn't need to be exactly the same as any of the ground truth answers, but should be semantically the same for the question. 2. Each item in the ground truth answer list can be viewed as a ground truth answer for the question, and the pred\_answer should be semantically the same as at least one of them.

question: question

ground truth answers: gt\_answer

pred\_answer: pred\_answer

The output should be in the following json format:

```
““json
{
  "rationale": "your rationale for the judgement, as a text",
  "judgement": "your judgement result, can only be 'correct'
or 'incorrect'"
}
””
```

Your output:

Table 9: Prompt for LLM-as-a-Judge.

## E Case Studies

In this section, we present comparative case studies derived from the interaction trajectories of **ReSearch** and our proposed **LeTS**. Specifically, Table 10 illustrates a scenario where **ReSearch** incorrectly answered a multi-hop question due to redundant and irrelevant search steps, whereas our proposed **LeTS** successfully provided the correct answer. This clearly demonstrates the effectiveness of integrating process-level rewards with outcome-level supervision.

---

**Question:** When did the president who set the precedent of a two term limit enter office?

**Golden Answer:** April 30, 1789

---

**ReSearch (Incorrect):** Misled by irrelevant searching leading to incorrect answer.

<think> To determine when the president who set the precedent of a two-term limit entered office, I need to first identify which president this is. I will use the Wikipedia search tool to find this information. </think>

<search> Who was the first president to set the two-term limit precedent? </search>

<result> "Term limits in the United States" Republican government as a periodic rotation". The historian Mercy Otis Warren, warned that ""there is no provision for a rotation, nor anything to prevent the perpetuity of office in the same hands for life; which by a little well-timed bribery, will probably be done"". Michael Korzi says ... </result>

<think> Now that I have the information, I can identify the president who set the precedent of a two-term limit. It was Thomas Jefferson. Now I need to find out when he entered office. </think>

<search> When did Thomas Jefferson enter office? </search>

<result> "Presidency of Thomas Jefferson" Presidency of Thomas Jefferson The presidency of Thomas Jefferson began on March 4, 1801, when he was inaugurated as the third President of the United States, and ended on March 4, 1809. Jefferson assumed the office after defeating incumbent President John Adams in the 1800 presidential election. The election was ... </result> (Irrelevant Document)

<think> Now I have the information I need. Thomas Jefferson entered office on March 4, 1801. </think>

<search> When did Thomas Jefferson become president? </search>

<result> "Presidency of Thomas Jefferson" Presidency of Thomas Jefferson The presidency of Thomas Jefferson began on March 4, 1801, when he was inaugurated as the third President of the United States, and ended on March 4, 1809. Jefferson assumed the office after defeating incumbent President John Adams in the 1800 presidential election. The election was ... </result> (Irrelevant and Redundant Documents)

<think> After reviewing the information, I can now provide the answer to the question. Thomas Jefferson, the president who set the precedent of a two-term limit, entered office on March 4, 1801. </think>

<answer> March 4, 1801 </answer>

---

**LeTS (Correct):** Effective, concise search leading to the correct answer.

<think> First, I need to identify which president set the precedent of a two-term limit. Then, I need to find out when that president entered office. Let's start by finding out which president established this precedent.

</think>

<search> which president established the two-term limit precedent </search>

<result> "Twenty-second Amendment to the United States Constitution" Twenty-second Amendment to the United States Constitution The Twenty-second Amendment (Amendment XXII) to the United States Constitution sets a limit on the number of times an individual is eligible for election to the office of President of the United States, and also sets additional eligibility conditions for presidents who succeed to the unexpired terms of their predecessors ... </result>

<think> From the information provided, it is clear that Franklin D. Roosevelt was the first president to break the two-term tradition by winning a third and fourth term. The precedent of a two-term limit was established by George Washington, who retired after two terms. The next step is to find out when George Washington entered office. </think>

<search> when did george washington enter office </search>

<result> "Presidency of George Washington" Presidency of George Washington The presidency of George Washington began on April 30, 1789, when Washington was inaugurated as the first President of the United States, and ended on March 4, 1797. Washington took office after the 1788–89 presidential election, the nation's first quadrennial presidential election, in which he was ... </result>

<think> The information clearly states that George Washington was inaugurated as the first President of the United States on April 30, 1789. This is the date when he entered office. </think>

<answer> April 30, 1789 </answer>

---

Table 10: Case study on the irrelevant searching phenomenon.