QAgent: A modular Search Agent with Interactive Query Understanding

Yi Jiang[*‡]**, Lei Shen**[*]**, Lujie Niu, Sendong Zhao**[†]**, Wenbo Su, Bo Zheng**
Taobao & Tmall Group of Alibaba
`jiangyijcx@163.com, zhaosendong@gmail.com`

Abstract

Large language models (LLMs) excel at natural language tasks but are limited by their static parametric knowledge, especially in knowledge-intensive task. Retrieval-augmented generation (RAG) mitigates this by integrating external information. However, (1) traditional RAG struggles with complex query understanding, and (2) even search agents trained with reinforcement learning (RL), despite their promise, still face generalization and deployment challenges. To address these limitations, we propose QAgent, a unified agentic RAG framework that employs a search agent for adaptive retrieval. This agent optimizes its understanding of the query through interactive reasoning and retrieval. To facilitate real-world application, we focus on modular search agent for query understanding that are plug-and-play in complex systems. Secifically, the agent follows a multi-step decision process trained with RL to maximize retrieval quality and support accurate downstream answers. We further analyze the strengths and weaknesses of end-to-end RL and propose a strategy that focuses on effective retrieval, thereby enhancing generalization in LLM applications. Experiments show QAgent excels at QA and serves as a plug-and-play module for real-world deployment. The code is available[1].

1 Introduction

Large language models (LLMs) have demonstrated outstanding performance across a wide range of natural language processing tasks(Jaech et al., 2024; Guo et al., 2025). However, for complex, knowledge-intensive tasks, LLMs face significant challenges, including outdated knowledge and hallucinations(Béchard and Ayala, 2024; Gade et al.,

---

[*]Equal contribution.
[†]Corresponding Author.
[‡]Work done during an internship at Alibaba
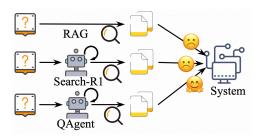[1]https://github.com/OpenStellarTeam/QAgent.git .

Figure 1: Different methods applied to the system.

2025), which undermine their accuracy and reliability. Retrieval-augmented generation (RAG) enables LLMs to access external knowledge by retrieving documents and using them as context for generation, demonstrating great potential in addressing these limitations(Lewis et al., 2020; Zhao et al., 2024a; Fan et al., 2024; Li et al., 2025b).

Nevertheless, the rigid workflow of traditional RAGs makes them inefficient for real-world problems requiring complex, multi-step reasoning. On the one hand, limitations exist in their ability to determine when and what to retrieve, making it difficult to construct accurate search paths for complex problems. On the other hand, RAGs are often embedded within complex systems, and the standalone RAG modules cannot adapt or optimize themselves according to the broader system's needs. To alleviate these limitations, recent work has proposed various solutions, including query optimization(Ma et al., 2023), planning(Jiang et al., 2025c), reflection(Asai et al., 2024), active retrieval(Jeong et al., 2024), and iterative retrieval(Trivedi et al., 2023). While these approaches have increased flexibility and reduced supervision costs, they still fail to effectively leverage feedback for continuous iteration and performance optimization. DeepSeek-R1(Guo et al., 2025) demonstrates that even rule-based, outcome-driven rewards can train powerful reasoning agents. This has led to the emergence of search agent trained with RL(Jin et al., 2025), enabling autonomous reasoning, on-demand retrieval,
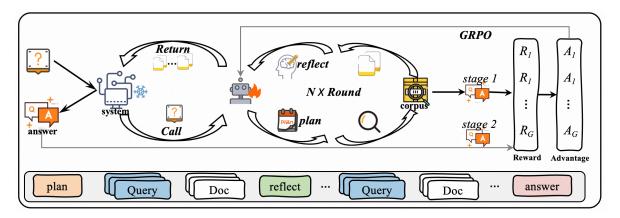
1

Figure 2: The proposed QAgent framework, with the left representing the system and the right representing our agent design, including "plan-search-information-reflect". In addition, the illustration shows a two-stage training framework, with the first stage using end-to-end RL and the second stage introducing generalized training. During training, the searched content tokens are masked to exclude them from loss calculations.

and iterative query optimization.

In addition, practical deployment of search agents remains challenging. As shown in Figure 1, on the one hand, due to query complexity, the information directly retrieved from the original query may not be useful; on the other hand, due to the complexity of the system, the search agent is often used as a submodule, aiming to improve its information retrieval capability rather than its information utilization capabilities.

From this, we identify two key goals for practical search agents: (1) understand and decompose complex queries to bridge the gap between the original query and the retriever; and (2) retrieve information that benefits the downstream generation, ensuring generalizability when deployed as a submodule.

To address these challenges, we propose a unified framework that integrates query understanding, reasoning, and iterative refinement into a coherent search process. The framework enables adaptive query decomposition through multiple rounds of interaction, where the model progressively refines its understanding of complex user intents by generating, executing, and reflecting on retrieval actions. At the core of the framework is a lightweight search agent capable of autonomous reasoning and decision-making. By formulating search as a sequential decision-making problem, the agent is trained end-to-end using reinforcement learning to maximize the quality of information returned to the downstream generation system. This allows the agent to learn not only what to retrieve, but also how to re-optimize in a context-aware manner, ultimately acting as a submodule to improve overall system performance.

Our contributions are summarized as follows:

- We propose a unified Agentic framework, centered around query understanding. Through multiple rounds of interaction, QAgent unifies query optimization, and continues to evolve through outcome reward feedback.

- We analyze the strengths and weaknesses of end-to-end reinforcement learning training and propose a two-stage training solution, providing profound insights for the practical application of search agents.

- We conduct extensive experiments demonstrate that QAgent delivers strong performance. And further analysis confirms its meaningful gains in information composition.

2 Related Work

2.1 Retrieval-Augmented Generation

Retrieval-augmented generation (RAG) has emerged as a promising approach to mitigate limitations of the LLMs, such as hallucination(Béchard and Ayala, 2024; Gade et al., 2025), by augmenting them with external retrieved knowledge(Lewis et al., 2020; Zhao et al., 2024a; Fan et al., 2024; Li et al., 2025b; Zhou et al., 2024a). Classic RAG systems typically follow a "retrieve-then-read" paradigm(Lewis et al., 2020), but it still faces numerous challenges that hinder optimal performance. To handle the complexity of problem, recent work identifies retrieval intent to skip unnecessary retrieval (Jeong et al., 2024; Cheng et al., 2024; Wang et al., 2023b) and refines

queries to better match retriever expectations (Ma et al., 2023; Wang et al., 2023a; Xu et al., 2024b). To counter noisy or irrelevant passages, methods introduce denoising(Wei et al., 2025), re-ranking(Glass et al., 2022; Xiao et al., 2024), and compression(Xu et al., 2024a; Jiang et al., 2023) between retrieval and generation. To bridge the retriever-generator gap, some jointly optimize both modules (Izacard et al., 2023; Lin et al., 2023) or insert trainable middlewares (Jiang et al., 2025b; Ke et al., 2024; Dong et al., 2025). Recent progress has increasingly moved towards agentic RAGs: Self-RAG(Asai et al., 2024) combines on-demand retrieval with reflection, MetaRAG(Zhou et al., 2024b) iteratively acquires knowledge through metacognition, and Search-o1(Li et al., 2025a) performs retrieval while reasoning. However, current RAG systems exhibit limited agentic capabilities and cannot evolve continuously, due to the lack of effective optimization methods.

2.2 Search Agent with RL

Reinforcement learning (RL)(Kaelbling et al., 1996) has emerged as a promising paradigm for improving the reasoning capabilities of large language models (LLMs). By interacting with the environment and driven by the goal of maximizing rewards, models are able to achieve autonomous reasoning and decision-making, such as OpenAI-o1(Jaech et al., 2024) and DeepSeek-R1(Guo et al., 2025). Recent research has further integrated RL into agent-based RAGs, enabling search agents to continuously learn and evolve through real-time interactions with search engines. For example, Search-R1(Jin et al., 2025), R1-Searcher(Song et al., 2025), Deepresearcher(Zheng et al., 2025) and ZeroSearch(Sun et al., 2025) utilize RL to train models for simultaneous search and reasoning, eliminating the need for supervision over intermediate reasoning steps. Subsequently, many improvements were made in reasoning patterns(Ren et al., 2025), reward verification(Wang et al., 2025; He et al., 2025) and efficiency(Jiang et al., 2025a; Sha et al., 2025). While these approaches have achieved impressive performance, their practical applicability remains unclear due to limitations in efficiency and generalization. In contrast, our work focuses on query understanding and analyzes training strategies that promote generalization, providing a solution for integrating search agents into complex systems in real-world applications.

3 Methodology

In this section, we introduce our agentic framework shown in Figure 2. It has two parts: the inference loop and the training strategy. The inference loop introduces our multi-round rollout for search agent. The training strategy first analyzes the RL training and then introduces a two-stage training.

3.1 Preliminaries

3.1.1 Agentic RAG

Naive RAG follows a retrieve-read paradigm. Let q be a query and $\mathcal{D}$ a document corpus. A retrieval function (including a retriever and a corpus) returns k relevant documents, denoted $\mathcal{R}(q)$. A generator $\mathcal{G}$ produces a answer $\hat{A}$ conditioned on q and $\mathcal{R}(q)$:

$$\hat{A} = \mathcal{G}(q, \mathcal{R}(q)). \tag{1}$$

In the agentic setting, search is modeled as a sequential process governed by policy $\pi_\theta(a_t \mid s_{<t})$, where $s_{<t}$ includes q, past actions, retrieved documents, and reasoning history. The agent selects actions $a_t \in \mathcal{A}$:

$$a_t \sim \pi_\theta(\cdot \mid s_{<t}), \tag{2}$$

typically reasoning or retrieval. The interaction yields a trajectory $\tau = (s_0, a_0, \ldots, s_T, a_T)$, with a final answer, framing RAG as a multi-step decision process for dynamic reasoning and retrieval.

3.1.2 Reinforcement Learning

Reinforcement learning (RL) is used to train the policy $\pi_\theta(y|x)$ by maximizing task-specific rewards $r_\phi(x, y)$. We adopt Group Relative Policy Optimization (GRPO)(Shao et al., 2024), which operates on G rollouts $\{y_i\}_{i=1}^G$ to compute group-relative advantages. The objective is:

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

where $\rho_{i,t} = \pi_\theta(y_{i,t}|x, y_{i,<t})/\pi_{\theta_{\text{old}}}(y_{i,t}|x, y_{i,<t})$, and $A_i = (R_i - \mu_R)/\sigma_R$ is the normalized advantage with μ_R and σ_R the mean and standard deviation of $\{R_j\}_{j=1}^G$. The KL term $\mathbb{D}_{\text{KL}} = \mathbb{D}_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_{\text{ref}}})$ constrains policy deviation. Our training is based on this algorithm.

3.2 Multi-Turn Query Optimization Loop

Motivation. One main goal of the search agent is to retrieve useful information. When the retriever and corpus are fixed, the way to achieve it is to optimize the query. Understanding the original query is complex(Zhao et al., 2024b), including but not limited to: the query requires multi-hop reasoning (Fig. 3(a)), needs to be decomposed (Fig. 3(b)), and does not conform to the retrieval system's preferences. Fully modeling this yields a wide and deep tree (Fig. 3(c)), which demands numerous model calls and retries, resulting in high complexity.
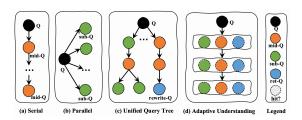

Figure 3: Illustration of query understanding.

Workflow. To address these challenges, inspired by recent advances in multi-round agentic RAG, we model the expansion of query understanding as a stochastic sequential decision process (Figure 3(d)). Instead of enforcing rigid transformation rules, our framework enables the agent to autonomously decide how to refine and issue queries over multiple interactions with a retrieval system.

Formally, given an initial query q, and conditioned on a fixed retriever $\mathcal{R}$ and retrieval corpus $\mathcal{D}$, the agent generates a sequence of optimized queries $S = \{S_1, \ldots, S_T\}$ over T rounds, where each $S_t = \{q_{t,1}, \ldots, q_{t,m_t}\}$ is a set of transformed queries produced at round t. The full trajectory probability is factorized (see Figure 2):

$$P(S \mid q, \mathcal{R}, \mathcal{D}) = \prod_{t=1}^{T} P\left(S_t \mid q, (I_i^{\text{pre}}, S_i, \mathcal{C}_i, I_i^{\text{post}})_{i<t}\right)$$
(4)

where $(\cdot)_{i<t}$ donates historical actions before round t, and I_i^{pre}, I_i^{post} encode pre-retrieval and post-retrieval reasoning traces (e.g., plan and reflection). The $\mathcal{C}_i$ donate the retrieval context, which aggregates results from all transformed queries:

$$C_i = \oplus_{j=1}^{m_i} \mathcal{R}(q_{ij}), \quad q_{ij} \in S_i,$$
(5)

where $\mathcal{R}(\cdot)$ fetches relevant documents from $\mathcal{D}$, and $\oplus$ represents context aggregation.

Overall, the agent follows a structured multi-round interaction loop, formalized as a trajectory:

$$\tau = (q, I_1^{pre}, S_1, \mathcal{C}_1, I_1^{post}, \ldots, \mathcal{C}_T, I_T^{post}, \hat{A}).$$
(6)

At each round t, the agent performs planning I_t^{pre}, generates optimized queries S_t. Followed by reflection I_t^{post}, in which the agent assesses the completeness of the accumulated information.

By unifying multiple query transformation paradigms within a stochastic, reflective interaction loop, our framework supports richer and more flexible query understanding than methods limited to fixed decomposition patterns, enabling the model to *ask better questions*.

3.3 End-to-End RL Training

For optimizing search agent, reinforcement learning is a powerful tool. It enables continuous interaction with the environment and uses outcome-based rewards for end-to-end optimization, thereby enhancing the LLM's agentic capability.

Reward Design. In the end-to-end RL training, our rewards are mainly designed based on two factors: the correctness of the answer and the correctness of the format. Formally,

$$R(\tau) = \mathbb{I}\{r_{\text{fmt}}(\tau) = 1\} \cdot \text{EM}_{\text{s}}(A^*, \hat{A}).$$
(7)

EM_{s} stands for Strict EM, the same as Search-R1. The reward format supports pre-retrieval planning, post-retrieval reflection, and final answer tags. This strict, sparse design may waste rollouts in early training, but ensures rollout quality and gradually improves efficiency as training stabilizes.

3.4 Generalized RL Training

Motivation. End-to-end training significantly improves the agent's capabilities, both in information retrieval and utilization. This training optimizes both retrieval and generation. However, due to the high latency of current autoregressive LLMs and the complexity of real-world systems, a single monolithic model is often insufficient to fully handle all tasks. We envision a search agent as a component that prioritizes information retrieval, while information utilization is not necessarily important. In this context, query optimization and information retrieval are our primary goals, and improving information utilization during training can become a form of reward hacking.

4

Figure 4: A case study of Agentic RL training (Search-R1), where both the fast ascent phase and the slow convergence phase are tested.

Analysis. We analyze a case study of the training process of an agent using reinforcement learning. As shown in Figure 4, in the early stages of model training, information retrieval and information utilization mutually enhance each other. However, later in training, the model tends to prioritize information utilization to obtain rewards. This undoubtedly compromises the generalization ability of the search agent as a submodule. An analysis of information utilization can be seen in § 4.5.

Generalized as Submodule. To address this, we introduced the second training stage. This involves using a frozen generator to answer based on the agent's retrieved documents, and then calculating rewards for responses from the frozen generator, thereby improving the agent's generalization as a tool. Formally,

$$\mathcal{K} = \text{parseDocSet}(\tau), \qquad (8)$$
$$\tilde{A} = \mathcal{G}(q, \mathcal{K}) \qquad (9)$$

where $\mathcal{K}$ is the retrieved passage set and $\tilde{A}$ is the frozen generator's predicted answer.

Reward Design. In this stage, constraints are shifted to the correctness of the generator's response, with format and mandatory answering requirements relaxed.

$$R(\tau) = \text{EM}(A^*, \tilde{A}) + 0.5 * Hit(\tau, A^*) \qquad (10)$$

EM is relaxed to non-strict EM because using strict EM on a frozen generator is inefficient, unlike a generator trained with a format reward. In addition, the reward for the model's end-to-end answer will be relaxed to Hit, meaning that the entire trajectory contains the gold answer.

Algorithm 1 Rollout with Multi-Turn Interaction

Require: Input query q, policy model π_θ, search engine $\mathcal{R}$, maximum turns B, Generator $\mathcal{G}$.
Ensure: Final predicted answer $\hat{A}$.
1: Initialization: rollout sequence $y \leftarrow \emptyset$, action count $b \leftarrow 0$, info set $\mathcal{K} \leftarrow \emptyset$
2: **while** $b < B$ **do**
3:     $y_b \leftarrow \emptyset$ ▷ Initialize current sequence
4:     **repeat**
5:        $y_t \sim \pi_\theta(\cdot \mid q, y + y_b), \quad y_b \leftarrow y_b \oplus y_t$
6:     **until** </search>, </answer>, <eos>in y_b
7:     $y \leftarrow y + y_b$
8:     **if** *SearchTags* detected in y_b **then**
9:        $Q \leftarrow \text{ParseQ}(S,\text{<query>}, \text{</query>})$
10:       $C = [\mathcal{R}(q_1), \cdots, \mathcal{R}(q_n)], q_i \in Q$
11:       $y \leftarrow y \oplus$ <information>C</information>
12:       $\mathcal{K} \leftarrow \mathcal{K} \cup C$ ▷ Update info set
13:     **else if** *AnswerTags* in y_b **then**
14:       **break**
15:     **else**
16:       Ask for retry: $y \leftarrow y \oplus prompt_{\text{rethink}}$
17:     **end if**
18:     Increment action count $b \leftarrow b + 1$
19: **end while**
20: $\tilde{A} = \mathcal{G}(q, \mathcal{K})$ ▷ Generate final answer
21: **return** final predicted answer $\tilde{A}$

3.5 Inference

For each query, QAgent starts with task instructions and reasons step by step. When it hits <search>...</search>tags, it extracts the query, retrieves documents, and appends them wrapped in <information>...</information>tags. This loop repeats, building a reasoning chain with external evidence. Finally, the passage set is fed to the generator to produce the answer (see Algorithm 1).

4 Experiment

This section presents the experimental results, and analysis. We main address the following questions: RQ1: How does QAgent perform in end-to-end QA?(§ 4.2) RQ2: How well does QAgent function as a submodule?(§ 4.2) RQ3: How does RL training affect information retrieval or usage?(§ 4.2 and § 4.5) RQ4: What advantages does the QAgent paradigm offer over traditional RAG?(§ 4.4)

Method	HotpotQA		2WikiMHQ†		MuSique†		NaturalQA		WebQ†		Average	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Search-o1	23.20	27.04	22.40	22.74	5.80	10.44	23.63	23.80	30.02	26.99	21.01	22.20
Search-R1	35.00	38.88	35.40	31.86	7.60	**14.01**	30.89	32.37	**35.68**	35.72	28.91	30.57
ZeroSearch	29.40	28.46	27.80	22.03	3.00	8.00	17.95	10.62	26.48	23.94	20.93	18.61
QAgent	**37.60**	**44.44**	**38.20**	**36.11**	**7.80**	12.39	**31.47**	**35.19**	32.09	**38.01**	**29.43**	**33.23**

Table 1: Main results of end-to-end performance. Best and second-best results are highlighted in **bold** and underlined, respectively. † denotes out-of-distribution evaluation datasets.

Method	HotpotQA		2WikiMHQ†		MuSique†		NaturalQA		WebQ†		Average	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Generator 3B												
Vanilla	15.00	19.33	23.00	25.01	0.80	6.40	17.53	19.58	28.44	29.38	16.95	19.94
NaiveRAG	30.60	35.54	25.40	28.32	3.20	7.64	30.22	31.43	29.97	30.26	23.88	26.64
Rerank	34.40	39.87	28.40	30.95	3.80	9.49	**35.35**	**35.20**	32.97	31.92	26.98	29.48
Search-o1	21.20	25.14	24.60	26.02	2.20	7.33	24.07	24.39	32.08	30.66	20.83	22.71
Search-R1	31.80	36.71	29.40	30.83	5.20	10.14	28.39	28.86	28.39	28.86	24.64	27.08
zero-search	26.40	30.46	24.80	26.07	2.00	4.98	19.31	20.40	23.13	24.74	21.17	24.86
QAgent	**37.00**	**42.94**	**33.80**	**35.22**	**5.20**	**10.20**	33.82	33.68	**36.32**	**32.78**	**29.23**	**30.96**
Generator 7B												
Vanilla	20.20	25.54	23.40	27.00	4.80	10.77	22.80	25.72	32.73	35.87	20.79	24.98
NaiveRAG	36.00	42.47	28.00	31.24	4.40	10.78	33.30	35.21	31.55	33.52	26.65	30.64
Rerank	39.00	44.76	28.40	32.11	5.40	11.84	**38.28**	**39.06**	33.66	34.75	28.95	32.50
Search-o1	27.00	31.15	24.40	27.78	6.00	11.77	28.42	30.06	33.91	35.40	23.95	27.23
Search-R1	36.00	40.63	34.40	33.41	7.00	12.35	31.27	32.09	32.07	33.94	28.15	30.48
ZeroSearch	28.40	34.44	24.00	27.10	3.00	7.47	23.21	25.00	27.26	30.27	21.17	24.86
QAgent	**42.40**	**46.75**	**35.80**	**36.76**	**7.40**	**13.81**	37.37	38.87	**36.71**	**36.41**	**31.94**	**34.52**

Table 2: Main results when used as a submodule. Best and second-best results are highlighted in **bold** and underlined, respectively. † denotes out-of-distribution evaluation datasets.

4.1 Experimental Setup

Datasets. We evaluate QAgent on five open-domain QA datasets covering both multi-hop and single-hop reasoning. The multi-hop QA benchmarks include **2WikiMultiHopQA**(Ho et al., 2020), **HotpotQA**(Yang et al., 2018) and **Musique**(Trivedi et al., 2022). For general QA, we use **WebQuestions**(Berant et al., 2013) and **NaturalQA**(Kwiatkowski et al., 2019). We adopt 500-sample subsets of 2WikiMultiHopQA and HotpotQA for efficiency, following Trivedi et al. (2023). More details are provided in Appendix A.

Evaluation. We report two metrics: EM and F1. Following prior work(Asai et al., 2024), we use a non-strict EM where a prediction is correct if it contains the gold answer. F1 measures token-level overlap between the predicted and gold answers. Since longer response may improve EM via coverage but introduce noise that lowers F1, evaluating both metrics allows for a more balanced evaluation.

Baselines. We compare QAgent with different baselines: (1) Vanilla and Naive RAG. Vanilla: direct answering without retrieval and Naive RAG: answering in the "retrieval-reading" paradigm. (2) Search-Agent. We utilize Search-o1, which uses multiple rounds of interaction but without RL training, and Search-R1 and zero-search, which use multiple rounds of interaction and joint retrieval and generation with RL training. (3) As the search agent component. We extract the retrieved documents from the model inference trajectory and input them into the frozen generators: Search-o1, Search-R1, zero-search, and the middleware Rerank. Details are in Appendix B

Implementation Details. We use Qwen-2.5-3B (Instruct)(Yang et al., 2025) as the training model and use frozen as the second-stage supervision model. For generation, we use Qwen-2.5-3B (Instruct) and 7B as generators. For retrieval, we use the 2018 Wikipedia dump as the knowledge source,

Method	HotpotQA		2WikiMHQ[†]		MuSique[†]		NQ		WebQ[†]		Average	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Generator 3B												
QAgent w/o all	31.20	37.03	29.20	30.81	4.40	9.67	28.95	29.63	30.36	29.87	24.82	27.40
+stage1	36.20	41.21	30.60	33.19	**5.40**	**10.24**	30.03	30.04	30.91	30.20	26.63	28.98
+stage1+stage2	**37.00**	**42.94**	**33.80**	**35.22**	5.20	10.20	**33.82**	**33.68**	**36.32**	**32.78**	**29.23**	**30.96**
Generator 7B												
QAgent w/o all	36.00	41.84	29.80	32.64	5.40	11.32	31.75	33.63	31.79	33.51	26.95	30.59
+stage1	40.40	45.84	35.20	37.65	7.40	13.60	33.52	34.93	32.58	33.83	29.82	33.17
+stage1+stage2	**42.40**	**46.75**	**35.80**	**36.76**	**7.40**	**13.81**	**37.37**	**38.87**	**36.71**	**36.41**	**31.94**	**34.52**

Table 3: Ablation Study. Best results are highlighted in **bold**. [†] denotes out-of-distribution evaluation datasets.

and all baselines use bm25(Robertson et al., 2009) as the retriever. For fairness, we retrieve 1 passage each query and deduplicate before sending to the generator. And the reported Search-R1 is a strictly aligned reproduction based on our framework.

4.2 Main Results

We present comprehensive evaluation results in Table 1 and Table 2, respectively. From the results, we have the following observations:

End-to-End QA performance. Overall, the RL-trained methods outperform the training-free methods, demonstrating the potential of RL training (the Zerosearch evaluation was not as good as expected, which we analyzed may be due to the sensitivity of the search engine). In an end-to-end QA comparison, our training achieved an improvement of 0.52% and 2.66% in EM and F1 respectively compared to Search-R1. Furthermore, we observed that on multi-hop datasets like Musique, our approach did not achieve significant improvement compared to Search-R1. We speculate that this may be due to our adaptive query optimization pattern. While it expands both the depth and breadth of query understanding, the inevitable expansion of breadth slightly compromises the depth.

Performance as Submodule. Table 2 shows the generalization of the search agent as a submodule in complex systems. We found that when freezing 3B as the generator, the overall performance of Search-R1 was significantly lower than that of end-to-end QA, indicating that its generalization was somewhat insufficient. In contrast, our method maintains strong generalization. Specifically, our approach improves average EM by 5.35% over NaiveRAG. It also outperforms end-to-end optimized search-R1 by 4.59% in average EM. And we found that the middleware rerank still achieves rela-

tively good performance, but it is a non-conflicting submodule with QAgent, which means they can be used simultaneously. Moreover, integrating our QAgent (3B) into a system with a 7B generator yields even greater gains, suggesting that complex systems can rely on a strong generator equipped with small search agent as submodule.

4.3 Ablation Study

In this section, we conduct detailed ablation experiments to verify the effectiveness of our training at each stage. As shown in the figure, our train-free version achieves high performance, with a significant improvement after end-to-end training. This benefited from the reinforcement learning algorithm, but end-to-end training still has shortcomings. We found that it significantly improved the performance in in-distribution evaluation, but the improvement was limited in out-of-distribution evaluation. After introducing a second stage of training, the performance was further improved, especially due to its improved generalization.
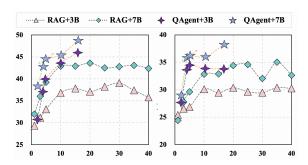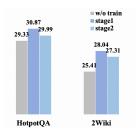

Figure 5: Illustration of the combined gain. The left is the experimental results of HotpotQA, and the right is that of 2WikiMHQ.

4.4 Combined Gain Analysis

In this section, we analyzed the advantages of using the search agent as a submodule. Specifically, we
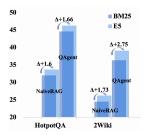
Figure 6: Information utilization ability

Figure 7: Different retrievers

increased the top-k threshold under Naive RAG to explore the upper limits of the "retrieval-reading" paradigm. As shown in Figure 5, due to the complexity of the problem, even with an increased top-k threshold, the gains gradually diminished. This is likely due to the redundancy of candidate document information and the limited amount of combinatorial information, which is an unavoidable limitation of all retrievers. The search agent, acting as an middleware between the query and the retriever, addresses the limitations of the retriever. Using the same token budget as the generator, it achieves even greater "combination gains", even exceeding the upper limits of the traditional paradigm.

Furthermore, we verified that using only the first few rounds significantly outperformed NaiveRAG, demonstrating its effectiveness. We also increased the retrieved top-k per query and found that performance continued to improve, but the number of documents used also increased significantly.

Overall, QAgent increases the upper limit of the model's performance when handling complex problems. However, in practical applications, a trade-off between the number of documents used and efficiency is necessary.

4.5 Information Utilization Analysis

As analyzed in Section 3.4, as the model training, its information utilization capability improves, and this is a obstacle to the model's information retrieval capability. Here, we analyze the two-stage trained models' ability to utilize the same information. Specifically, we perform naive RAG on the trained models, give them the same top-1 document, and then test the performance of the responses. We found that the end-to-end trained model achieved the highest performance. This is because information utilization capability is the direct goal of training. After generalization training, the model's information utilization capability decreased. We speculate that this is because the

optimization goal at this time is no longer focused on information utilization, but gradually focuses on information retrieval capability. This provides new insights into the training of search agents.

4.6 Generalization to Different Retrievers

In this section, we analyzed the generalization of QAgent. As shown in Figure 7, we used two representative search engines, BM25 and E5. As expected, using the more powerful search engine, E5, yielded better performance. Specifically, the strong retriever enables QAgent to achieve improvements on HotpotQA similar to Naive RAG. However, powerful search engine yielded a greater performance improvement on 2WikiMHQ compared to Naive RAG. This suggests that the QAgent middleware is plug-and-play, compatible with both search engines and generators, and exhibiting high generalization performance.
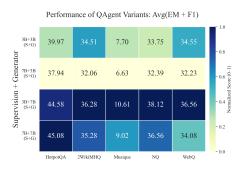

Figure 8: The performance difference between using different generators during training and inference.

4.7 Different Model Sizes for Supervision

Using the generator's responses to indirectly calculate the reward, rather than relying on end-to-end answer, helps avoid reward hacking that may arise from over-optimizing information utilization. However, the impact of different generators for supervision requires further analysis. We experimented with two generator sizes during training: 3B and 7B. As shown in Figure 8, training with the 7B generator led to more stable, but 3B achieved better overall performance. We speculate that this results from the 3B model's inherent instability, which acts as a form of regularization. Specifically, if a smaller generator can produce correct answers, it suggests that the retrieved information is sufficient for accurate reasoning.

During inference, larger generators consistently yield better performance. This indicates that complex systems benefit from more powerful genera-

tors, and a lightweight search agent can be effectively integrated to meet efficiency.

5 Conclusion

In this work, we present QAgent, a unified agentic RAG framework that tackles two key challenges: weak complex query understanding and limited generalization of search agents. QAgent employs a modular, plug and play agent that optimizes queries through iterative reasoning and retrieval, trained with RL to maximize retrieval quality. Experiments show QAgent outperforms existing methods on complex QA tasks.

6 Limitation

While QAgent achieves strong performance, several limitations remain.

Training on larger models. Our motivation is to serve as a submodule of the system. Within this goal, validation on smaller models is paramount. However, whether larger models achieve consistent results remains to be verified.

Failure to control passage diversity. We tried various approaches to passage diversity, including designing repeated negative rewards and adjusting passage overlap, but found that these all led to severe reward hacking, especially when the number of passages is uncertain. Diversity and number are strongly correlated, and the model may overreact by increasing the number of irrelevant passages.

Robustness challenges. The retriever is a factor that influences training and inference, affecting the agent's learning patterns, particularly re-retrieval. Furthermore, intuitively, query understanding and search engine preferences are relatively consistent. For example, sparse retriever tend to favor keywords. Although we have verified that QAgent achieves robust performance on different search engines, it remains a challenge to directly achieve a query optimization goal that satisfies the preferences of all search engines.

References

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

Patrice Béchard and Orlando Marquez Ayala. 2024. Reducing hallucination in structured outputs via retrieval-augmented generation. *arXiv preprint arXiv:2404.08189*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544.

Qinyuan Cheng, Xiaonan Li, Shimin Li, Qin Zhu, Zhangyue Yin, Yunfan Shao, Linyang Li, Tianxiang Sun, Hang Yan, and Xipeng Qiu. 2024. Unified active retrieval for retrieval augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 17153–17166.

Guanting Dong, Yutao Zhu, Chenghao Zhang, Zechen Wang, Ji-Rong Wen, and Zhicheng Dou. 2025. Understand what llm needs: Dual preference alignment for retrieval-augmented generation. In *Proceedings of the ACM on Web Conference 2025*, pages 4206–4225.

Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pages 6491–6501.

Anoushka Gade, Jorjeta G Jetcheva, and Hardi Trivedi. 2025. It's about time: Incorporating temporality in retrieval augmented language models. In *2025 IEEE Conference on Artificial Intelligence (CAI)*, pages 75–82. IEEE.

Michael Glass, Gaetano Rossiello, Md Faisal Mahbub Chowdhury, Ankita Naik, Pengshan Cai, and Alfio Gliozzo. 2022. Re2g: Retrieve, rerank, generate. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2701–2715.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Jie He, Víctor Gutiérrez Basulto, and Jeff Z Pan. 2025. From sufficiency to reflection: Reinforcement-guided thinking quality in retrieval-augmented reasoning for llms. *arXiv preprint arXiv:2507.22716*.

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.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2023. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43.

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*.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. 2024. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7029–7043.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*.

Pengcheng Jiang, Xueqiang Xu, Jiacheng Lin, Jinfeng Xiao, Zifeng Wang, Jimeng Sun, and Jiawei Han. 2025a. s3: You don't need that much data to train a search agent via rl. *arXiv preprint arXiv:2505.14146*.

Yi Jiang, Sendong Zhao, Jianbo Li, Haochun Wang, and Bing Qin. 2025b. GainRAG: Preference alignment in retrieval-augmented generation through gain signal synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10746–10757. Association for Computational Linguistics.

Zhouyu Jiang, Mengshu Sun, Lei Liang, and Zhiqiang Zhang. 2025c. Retrieve, summarize, plan: Advancing multi-hop question answering with an iterative approach. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 1677–1686.

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*.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

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

Zixuan Ke, Weize Kong, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. 2024. Bridging the preference gap between retrievers and llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10438–10451.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, and 1 others. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

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.

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

Xiaoxi Li, Jiajie Jin, Yujia Zhou, Yuyao Zhang, Peitian Zhang, Yutao Zhu, and Zhicheng Dou. 2025b. From matching to generation: A survey on generative information retrieval. *ACM Transactions on Information Systems*, 43(3):1–62.

Xi Victoria Lin, Xilun Chen, Mingda Chen, Weijia Shi, Maria Lomeli, Richard James, Pedro Rodriguez, Jacob Kahn, Gergely Szilvasy, Mike Lewis, and 1 others. 2023. Ra-dit: Retrieval-augmented dual instruction tuning. In *The Twelfth International Conference on Learning Representations*.

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.

Jingyi Ren, Yekun Xu, Xiaolong Wang, Weitao Li, Weizhi Ma, and Yang Liu. 2025. Effective and transparent rag: Adaptive-reward reinforcement learning for decision traceability. *arXiv preprint arXiv:2505.13258*.

Stephen Robertson, Hugo Zaragoza, and 1 others. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Zeyang Sha, Shiwen Cui, and Weiqiang Wang. 2025. Sem: Reinforcement learning for search-efficient large language models. *arXiv preprint arXiv:2505.07903*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan

Zhang, YK Li, Yang 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, Lei Fang, and Ji-Rong Wen. 2025. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*.

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. 2025. Zerosearch: Incentivize the search capability of llms without searching. *arXiv preprint arXiv:2505.04588*.

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

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

Liang Wang, Nan Yang, and Furu Wei. 2023a. Query2doc: Query expansion with large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9414–9423.

Yile Wang, Peng Li, Maosong Sun, and Yang Liu. 2023b. Self-knowledge guided retrieval augmentation for large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10303–10315.

Ziliang Wang, Xuhui Zheng, Kang An, Cijun Ouyang, Jialu Cai, Yuhang Wang, and Yichao Wu. 2025. Stepsearch: Igniting llms search ability via stepwise proximal policy optimization. *arXiv preprint arXiv:2505.15107*.

Zhepei Wei, Wei-Lin Chen, and Yu Meng. 2025. InstructRAG: Instructing retrieval-augmented generation via self-synthesized rationales. In *The Thirteenth International Conference on Learning Representations*.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pages 641–649.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024a. Recomp: Improving retrieval-augmented lms with compression and selective augmentation. In *12th International Conference on Learning Representations, ICLR 2024*.

Shicheng Xu, Liang Pang, Huawei Shen, Xueqi Cheng, and Tat-Seng Chua. 2024b. Search-in-the-chain: Interactively enhancing large language models with search for knowledge-intensive tasks. In *Proceedings of the ACM Web Conference 2024*, pages 1362–1373.

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 23 others. 2025. Qwen2.5 technical report. *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.

Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, Jie Jiang, and Bin Cui. 2024a. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473*.

Siyun Zhao, Yuqing Yang, Zilong Wang, Zhiyuan He, Luna K Qiu, and Lili Qiu. 2024b. Retrieval augmented generation (rag) and beyond: A comprehensive survey on how to make your llms use external data more wisely. *arXiv preprint arXiv:2409.14924*.

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*.

Yujia Zhou, Yan Liu, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Zheng Liu, Chaozhuo Li, Zhicheng Dou, Tsung-Yi Ho, and Philip S Yu. 2024a. Trustworthiness in retrieval-augmented generation systems: A survey. *arXiv preprint arXiv:2409.10102*.

Yujia Zhou, Zheng Liu, Jiajie Jin, Jian-Yun Nie, and Zhicheng Dou. 2024b. Metacognitive retrieval-augmented large language models. In *Proceedings of the ACM Web Conference 2024*, pages 1453–1463.

A Dataset

Here, we provide a detailed description of the datasets we used.

HotpotQA(Yang et al., 2018) and **2WikiMultiHopQA**(Ho et al., 2020): Both datasets are multi-hop question answering datasets based on Wikipedia. Given the cost constraints of the experiment, we used a subsample of the Trivedi et al. (2023) dataset, which was obtained by extracting 500 questions from the validation set of each dataset.

Musique(Trivedi et al., 2022): By iteratively selecting composable single-hop question pairs, we created questions spanning 2-4 hops, which are generally more difficult.

WebQuestions(Berant et al., 2013): Constructed from questions posed by the Google Suggest API, where the answers are specific entities listed in Freebase.

NaturalQA(Kwiatkowski et al., 2019): A dataset designed to support comprehensive question answering systems. It contains questions from real Google search queries, and the corresponding answers are text snippets from Wikipedia articles, carefully identified by human annotators.

B baselines

We compare QAgent with different baselines:
Vanilla: Direct answering without retrieval.

Naive RAG: Predicting an answer in the "retrieval-then-reading" paradigm.

Search-o1: Using multiple rounds of interaction but without RL training.

Search-R1: Using multiple rounds of interaction and joint retrieval and generation model with RL training.

zero-search: Training a LLM as retriever can reduce the cost and enhance the robustness to the retriever.

Rerank: Follows the "retrieve-rearrange-generate" paradigm and is the most widely used middleware (this does not conflict with QAgent).

When as the search agent component, we extract the retrieved documents from the model inference trajectory and input them into the frozen generators.

C Implementation Details

We experimented with Qwen-2.5-3B (Instruct)(Yang et al., 2025). For retrieval, we used the 2018 Wikipedia dump(Karpukhin et al., 2020) as the knowledge source and BM25(Robertson et al., 2009) as the retriever. To ensure a fair comparison, we set the number of retrieved passages to 5 for all retrieval-based methods and 1 for iterative methods. For training, we followed Search-R1, merging the training sets of NQ(Kwiatkowski et al., 2019) and HotpotQA(Yang et al., 2018) to form a unified dataset.

For GRPO training, we set the learning rate of the policy LLM to 1e-6. The KL divergence regularization coefficient β and the clipping rate ϵ were

set to 0.001 and 0.2, respectively. We sampled 5 responses for each prompt. The maximum sequence length was set to 8192 tokens, the maximum response length was 512. To accelerate sampling, we use vLLM[2] as the inference acceleration framework. We set the rollout temperature and top-p value of our vLLM-based rollout to 1.0. A purely online algorithm was used, meaning that the vLLM weights used for sampling were updated after each weight update. Training is performed on a single node equipped with 8 GPUs, four of which are used for running rollout, three for training, and one for deploying the reference model. For training, we used Liger-Kernel[3] to optimize training efficiency. The actual batch size for a single GPU is 1, the gradient accumulation is set to 64, and training is performed until 1400 actual update steps, with a warm-up step number of 225. In addition, we took a checkpoint of 1120 steps for stage 1 training, and a checkpoint of 1400 steps for stage 2. During the evaluation, we used the vllm framework and performed inference on a single GPU.

D Training

Figure 9 shows the stability of the reward increase. Compared with Search-R1, our method has a higher starting point and a higher upper limit of convergence. As shown in Figure 10, In the second stage of training, we used different frozen generators for supervision, and we can see that the reward steadily increased. Using 7B supervision is more stable and has higher rewards, because the larger generator has richer internal knowledge. However, from the experimental evaluation results, 3B has slightly better generalization.
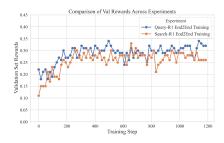

Figure 9: Illustration of the reward increasing during end-to-end training.

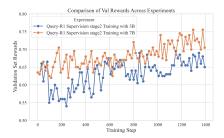
Figure 10: Illustration of the reward increasing during stage2 training. Note that the upper limits of the rewards is different.

E   Experimental Results

We present here the complete experimental results of Section 4.7, as shown in Table 4.

F   Template Prompt

We present here the instruction prompt in Table 5.

G   Case Study

A case of QAgent is shown in Figure 11.

Method	HotpotQA		2WikiMHQ[†]		MuSique[†]		NaturalQA		WebQuestion[†]		Average	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Generator 3B												
End-to-End	36.20	41.21	30.60	33.19	5.40	10.24	30.03	30.04	30.91	30.20	26.63	28.98
Supervision: 3B	37.00	42.94	33.80	35.22	5.20	10.20	33.82	33.68	36.32	32.78	29.23	30.96
Supervision: 7B	34.80	41.07	31.40	32.72	4.20	9.07	32.49	32.30	32.73	31.74	27.12	29.38
Generator 7B												
End-to-End	40.40	45.84	35.20	37.65	7.40	13.60	33.52	34.93	32.58	33.83	29.82	33.17
Supervision: 3B	42.40	46.75	35.80	36.76	7.40	13.81	37.37	38.87	36.71	36.41	31.94	34.52
Supervision: 7B	42.00	48.16	33.80	36.77	6.00	12.04	35.87	37.26	33.51	34.65	30.24	33.78

Table 4: Full experimental results using different models as supervision.

Search for information to answer the given question. You can search as many times as needed if you find you lack some knowledge. You will go through a loop of:
"<plan>xxx</plan>
<search >xxx</search>
<information>xxx</information>
<reflection>xxx </reflection>
<plan>xxx</plan>(if not complete)
    ...
<reflection>xxx </reflection>
<answer>xxx</answer>".
You must conduct planning inside <plan>and </plan>first every time you call a search engine.
After planing, you can call a search engine to search multiple queries (no more than 3) by
<search>
    <query>query1</query>
    <query>query2</query>
    ...
    <query>queryk</query>
</search>,
and it will return the searched results between <information>and </information>.
After getting information, you must conduct a reflection on the information and place your reflection between the <reflection>and </reflection>tags.
Note that you must plan within <plan>and </plan>before searching, and reflect within <reflect>and </reflect>after receiving information.
Note that each query must be enclosed between <query>and </query>, and all queries must be placed between <search>and </search>, such as <search>\n <query>query1</query>\n <query>query2</query>\n ... \n <query>queryk</query>\n </search>.
If the task is not yet complete, begin a new <plan>.
If you find no further external knowledge needed, you can directly provide the answer inside <answer>and </answer>without detailed illustrations.
For example, <answer>\n xxx\n </answer>.
The answer, "xxx", should be a few short words.
Question: {$question$}.

Table 5: Template prompt used in experiments.

Query: Baraki Barak District is situated in the western part of a province whose capital is what?

Gold Answer: Puli Alam

Search Path:
<plan> I need to find the province where Baraki Barak District is situated and then determine the capital of that province. </plan>
<search>
<query>Baraki Barak District province</query>
<query>Baraki Barak District capital province</query>
<query>location of Baraki Barak District</query>
</search>
<plan>I need to find the capital of Logar Province.</plan>
<search>
<query>capital of Logar Province, Afghanistan</query>
<query>Logar Province capital city</query>
<query>province capital Logar Province</query>
</search>
<plan> Based on the information found, I can now provide the answer to the question.</plan>
<answer> Puli Alam </answer>

Figure 11: A case of QAgent.