Efficient Search Agent via Atomic Query and Memory Purification

Anonymous ACL submission

Abstract

Retrieval-augmented generation (RAG) mitigates hallucinations in large language models (LLMs) by conditioning generation on retrieved external knowledge. Recent search agents extend RAG into an autonomous, multi-turn information-seeking process, yet existing methods tend to accumulate irrelevant documents and rely on sparse reinforcement learning signals. We propose **SE-Search**, an efficient search agent that co-optimizes querying, evidence filtering, and answer generation within a single policy π_θ through dense rewards. Three mechanisms drive this co-optimization: memory purification filters retrieved noise into a concise evidence state, atomic query training encourages short and diverse subqueries, and a four-component dense reward provides functionally distinct feedback on query, memory, answer, and format dimensions. Unlike prior single-policy agents that rely solely on sparse final-answer rewards, SE-Search enables efficient end-to-end co-optimization under shared GRPO gradients. Experiments on seven single-hop and multi-hop QA benchmarks show that `SE-Search-3B` achieves the highest overall accuracy among compared methods, yielding a 10.8-point absolute improvement and a 34.6% relative gain over Search-R1-3B while using fewer search steps. The code is available at `https://anonymous.4open.science/r/SE-Search-6120`.

1 Introduction

Information seeking (Case and Given, 2016) is a fundamental component of knowledge work. Web search engines such as Google and Bing return ranked pages for a given query, but as queries grow in complexity, traditional information retrieval techniques (Kobayashi and Takeda, 2000) increasingly struggle to capture nuanced user intent and may return results that are only tangentially related to the question.

Large language models (LLMs) (Zhao et al., 2023) exhibit strong capabilities in language understanding, reasoning, and information integration, yet they lack reliable access to up-to-date external knowledge and can produce factual errors. Retrieval-augmented generation (RAG) (Lewis et al., 2020) mitigates these issues by conditioning generation on retrieved passages. However, fixed RAG pipelines do not allow the LLM to decide *when* to search or *what* to search for, limiting their flexibility on complex real-world queries.

Recent work (Li et al., 2025a) introduces LLM-based agents (Guo et al., 2024) that interpret user intent, plan search strategies, execute multi-turn retrieval actions, and synthesize the retrieved evidence. These systems, commonly referred to as search agents, bridge language understanding with active information retrieval. Representative methods such as Search-R1 (Jin et al., 2025) train agents via reinforcement learning (RL) (Kaelbling et al., 1996; Shao et al., 2024) with rule-based rewards to interact with a search tool over a large corpus.

Despite promising results, existing search agents face three challenges that hinder efficient multi-hop retrieval. **Noisy search results:** Top-K retrieval often returns irrelevant documents that dilute useful context. MAIN-RAG (Chang et al., 2025) addresses this via multi-LLM filtering, but operates in a training-free setting. **Limited query diversity:** Prior methods (Shi et al., 2025a) tend to generate similar queries across steps, restricting evidence breadth. O^2-Searcher (Mei et al., 2025) introduces a diversity reward based on embedding similarity, but the extra encoding step is costly. **Sparse feedback:** Search-R1 supervises only final-answer correctness without rewarding query quality or search frequency, causing agents to produce overly long queries regardless of question complexity. TooRL (Qian et al., 2025) proposes richer rewards for general tool use, but does not target search-specific behavior.

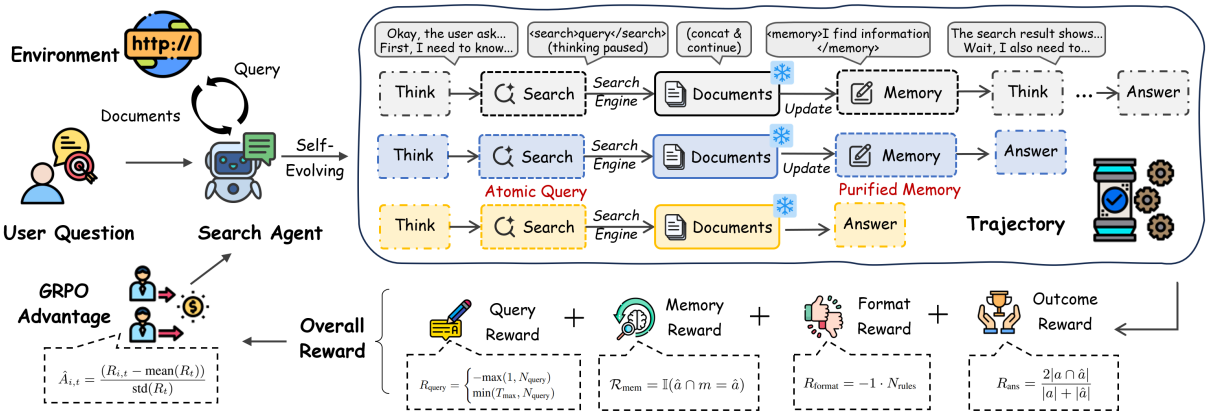

Figure 1: Training scheme of SE-Search. For each question, the search agent generates diverse trajectories comprising the steps *think*, *search*, *memorize*, and *answer*. These trajectories are optimized using the GRPO algorithm (Shao et al., 2024) and four carefully designed rewards: Query, Format, Memory, and Outcome.

We propose SE-Search, an efficient search agent that improves its search behavior through RL-driven optimization with structured feedback. Rather than employing explicit iterative data-generation–retraining cycles as in EvolveSearch (Zhang et al., 2025a), SE-Search achieves progressive policy improvement within a single RL training phase, where the agent's own interaction experience and structured dense rewards continuously reshape its search strategy (Ruse, 1975; ang Gao et al.). SE-Search introduces three co-optimized mechanisms. (1) **Memory Purification** adopts a *memorize-after-search* paradigm that filters retrieved documents into a concise memory state. While Memory-R1 (Yan et al., 2025) separates memory and answer generation into two distinct agents, and DeepAgent (Li et al., 2025d) relies on auxiliary LLMs for memory management, SE-Search performs both within a *single* policy π_θ, where memory writing and answer generation share the same GRPO gradient signal—eliminating inter-agent communication overhead and enabling end-to-end co-optimization. (2) **Atomic Query**, motivated by Atom-Searcher (Deng et al., 2025), enables the agent to autonomously learn concise, non-redundant query decomposition via a character-level substring non-overlap penalty (Eq. 3), integrating query optimization directly into π_θ without auxiliary rewriters. (3) **Dense Rewards** provide four fine-grained feedback signals—Query, Memory, Outcome, and Format—each grounded in a specific term of the formal optimization objective (Eq. 2), providing targeted feedback on different dimensions of search behavior. Unlike GIGPO (Feng et al., 2025), which collapses intermediate rewards into a single discounted scalar, our four terms are functionally distinct rule-based proxies that require no auxiliary model inference.

Our contributions are threefold:

- We propose SE-Search, an efficient search agent that co-optimizes memory management, query formulation, and answer generation within one policy π_θ via dense rewards—unlike prior agents that rely on sparse final-answer signals alone.
- We design three reward-driven mechanisms—Memory Purification via CEM-based supervision, Atomic Query via a substring non-overlap penalty, and a four-component Dense Reward grounded in the formal objective (Eq. 2)—each targeting a distinct dimension of search behavior without auxiliary model inference.
- Experiments on seven QA benchmarks show that SE-Search-3B outperforms recent baselines efficiently: achieving higher EM at fewer search steps and lower computational cost.

2 Problem Formulation

We model the question answering task as $a = \pi_\theta(Q)$, where Q is the user question, a is the generated answer, and π_θ denotes the LLM. In retrieval augmented generation, the system retrieves knowledge k from an external corpus $\mathcal{D}$ using a retriever $\mathcal{R}(\cdot)$ and generates the answer conditioned on both: $a = \pi_\theta(Q \mid k),\ k = \mathcal{R}(Q \mid \mathcal{D})$.

However, this fixed workflow is insufficient for complex questions that require iterative reasoning and adaptive retrieval, because the number of retrieval steps, the content of each query, and the order in which evidence is gathered all affect the final answer quality. Search agents address this by interleaving retrieval with reasoning within a

Method	Architecture	Mem. Reward	Query Reward
Search-R1	Single	✗	✗
Memory-R1	Multi-agent	✗	✗
DeepAgent	Aux. LLM	✗	✗
AutoRefine	Single	✗	✗
O^2-Searcher	Single	✗	Embed. sim.
SE-Search (Ours)	**Single**	**CEM**	**Substr. penalty**

Table 1: Design comparison with representative search agent methods. SE-Search jointly optimizes memory retention and query diversity through dedicated dense rewards under a single policy.

dynamic, multi-turn loop. In this setting, the model output y may include both intermediate reasoning steps r and the final answer a.

$$a = \pi_\theta\left(Q \mid \{r_t \otimes k_t\}_{t=1}^{T}\right), k_t = \mathcal{R}(q_t \mid \mathcal{D}), q_t \in r_t \quad (1)$$

where q_t denotes subqueries generated during reasoning to retrieve intermediate knowledge k_t, and $\otimes$ denotes the integration of retrieved knowledge into the reasoning trajectory. When only final-answer supervision is available, one can apply supervised fine-tuning (SFT) to maximize $\max_\theta \mathbb{E}[\log \pi_\theta(\hat{a})]$, but this objective ignores the quality of intermediate search queries and retrieved evidence.

3 Approach

We present `SE-Search`, a search agent that improves its search behavior through RL-driven optimization without auxiliary LLMs or multi-agent coordination. The three components are co-optimized through a single policy π_θ under GRPO, forming a virtuous cycle where each component's improvement benefits the others. Table 1 summarizes the architectural differences with prior methods. We first formalize the optimization objective and then describe each component.

3.1 Search Agent Formulation

Figure 1 illustrates the optimization scheme of SE-Search. Given a user question, the agent interacts with a retrieval environment to generate multiple trajectories τ. Each trajectory $\tau = (\tau_1, \tau_2, \ldots, \tau_T)$ consists of steps $\tau_t = (a_t, c_t)$, where the action $a_t \in$ {<think>, <search>, <documents>, <memory>, <answer>} and the content c_t is a reasoning thought, a search query, retrieved passages, a distilled memory summary, or a final answer. The key challenge is that the agent must learn *what* to search, *when* to stop searching, and *how* to distill evidence—all

> You are a capable reasoning assistant, able to perform multiple search calls and memory purification to answer questions. You must reason through the available information using <think> and </think>. If you lack knowledge, you can call a search engine using <search> query </search> and it will return the top three results between <documents> and </documents>. After each search, extract useful information from these documents and supplement or revise your memory between <memory> and </memory>. You may send multiple search requests if needed. Do not repeat search queries. Once you have sufficient information, provide a concise final answer using <answer> and </answer>. For example, <answer> Donald Trump </answer>. Question {**question**}

Figure 2: Prompt template for SE-Search.

from reward signals alone, without access to ground-truth trajectories.

Formally, we optimize $\max_\theta \mathbb{E}[\log \pi_\theta(\hat{a}) + \alpha \cdot \mathcal{C}(\cup_{t=1}^{T} k_t, \hat{K})]$, where $\mathcal{C}(\cdot,\cdot)$ measures the coverage of retrieved documents relative to the expected knowledge $\hat{K}$, T is the number of retrieval steps, and $\alpha \in (0,1)$ is a weighting factor. Since Search-R1 (Jin et al., 2025) shows that loss masking can exclude retrieved tokens from gradient computation, we approximate the coverage term using the search queries and memory contents:

$$\max_\theta \mathbb{E}\Big[\log \pi_\theta(\hat{a}) + \alpha \cdot \sum_t \log \pi_\theta(\hat{m}_t) + \gamma \cdot \sum_t \log \pi_\theta(\hat{q}_t)\Big] \quad (2)$$

where γ is an additional weighting factor. This objective jointly optimizes three aspects of agent behavior: query formulation, memory construction, and final-answer generation. Because ground-truth trajectories are unavailable, supervised fine-tuning on intermediate steps is not feasible. We therefore adopt a post-training RL framework with rule-based reward functions to optimize trajectory generation.

3.2 Memory Purification

Prior methods pass all retrieved documents directly to the LLM without filtering, forcing the model to reason over a mixture of relevant and irrelevant content. We introduce memory purification to filter and consolidate relevant facts from retrieved documents using the special token <memory>. While Memory-R1 (Yan et al., 2025) employs a dual-agent framework with separate memory and answer agents, and DeepAgent (Li et al., 2025d), AgentFold (Ye et al., 2025), and MAIN-RAG (Chang et al., 2025) rely on auxiliary LLMs or external modules for context management, SE-Search performs memory management and answer generation within a *single* policy π_θ. This design means that memory writing and

answer generation are jointly optimized under the same GRPO gradient—R_{mem} shapes how the agent distills evidence and R_{ans} shapes how it produces answers—eliminating inter-agent communication and auxiliary model dependencies while enabling end-to-end co-optimization.

Figure 2 shows the prompt that instructs the LLM to update memory by incorporating retrieved knowledge k via $m_t = \pi_\theta(m_{t-1} \mid k_t)$, selectively retaining answer-relevant facts to maintain a progressively refined knowledge state across turns. To directly supervise this retention—unlike prior search agents (Jin et al., 2025; Chen et al., 2025) that rely solely on final-answer correctness—we define a memory reward using Cover Exact Match (CEM). While AutoRefine (Shi et al., 2025a) applies CEM-based reward to train document refinement (compressing retrieved passages), SE-Search applies it to train the agent's *memory writing* decisions—directly optimizing which evidence to retain across turns. Following AutoRefine (Shi et al., 2025a), the memory reward is defined as $R_{\text{mem}} = \mathbb{I}(\hat{a} \cap m = \hat{a})$, where $\mathbb{I}(\cdot)$ is the indicator function and CEM checks whether the memory m contains all tokens of the ground-truth answer $\hat{a}$. Trivial reward hacking (e.g., copying entire documents) is prevented by the format reward R_{format}, which penalizes excessive output length, and by the finite context window that physically constrains memory size (empirically, memory segments average 45 tokens and remain below 120 tokens).

3.3 Atomic Query

Query decomposition and rewriting have been extensively studied in information retrieval. However, prior approaches (Ma et al., 2023) typically employ auxiliary LLMs to reformulate or decompose queries in a separate pipeline stage, which introduces additional inference cost and decouples query optimization from downstream answer generation. In contrast, SE-Search integrates efficient atomic query generation into the *same single policy* π_θ used for reasoning, memory management, and answer generation. The agent autonomously learns to decompose complex questions into short, focused subqueries across multiple search turns, guided solely by RL reward signals. This end-to-end design ensures that query quality directly influences—and is influenced by—memory purification and answer generation through shared GRPO gradients. In Algorithm 1, we propose an atomic query counting method

Algorithm 1: Atomic Query Counting

Input: Search query set $\mathcal{Q}$, length bounds $[L_{\min}, L_{\max}]$, similarity threshold T.
Output: Number of valid atomic queries N_{query}.

Initialize valid query set $\mathcal{Q}_v \leftarrow \varnothing$;
for $q_i \in \mathcal{Q}$ **do**
  if $L_{\min} \leq |q_i| \leq L_{\max}$ **then**
    if $\mathcal{Q}_v = \varnothing$ **then**
      $\mathcal{Q}_v = \{q_i\}$; **continue**;
    end if
    valid $\leftarrow$ True;
    for $q_j \in \mathcal{Q}_v$ **do**
      Compute matching blocks $\{\ell_k\}$ via SequenceMatch(q_i, q_j);
      $$\text{ratio}(q_i, q_j) = \frac{2\sum_k \ell_k}{|q_i| + |q_j|} \quad (3)$$
      if $ratio(q_i, q_j) > T$ **then**
        valid $\leftarrow$ False; **break**;
      end if
    end for
    if *valid* **then**
      $\mathcal{Q}_v = \mathcal{Q}_v \cup \{q_i\}$;
    end if
  end if
end for
Return $N_{query} \leftarrow |\mathcal{Q}_v|$;

that constrains query length and enforces diversity via a character-level substring non-overlap penalty (Eq. 3). We adopt character-level matching rather than embedding-based similarity because it requires no auxiliary encoder inference and, in factoid QA, equivalent reformulations typically share substantial lexical overlap. This enables the agent to learn decomposition strategies through GRPO rather than handcrafted rules. Based on the number of valid atomic queries N_{query} in a trajectory, we define a query reward that guides when and how the agent invokes the search tool.

$$R_{\text{query}} = \begin{cases} -\max(1, N_{\text{query}}) & \text{if the answer is correct} \\ \min(T_{\max}, N_{\text{query}}) & \text{if the answer is incorrect} \end{cases} \quad (4)$$

where $T_{\max}$ is the maximum allowed number of search action turns, and N_{query} denotes the number of valid search queries in the trajectory. This reward serves two complementary purposes: it encourages the agent to issue diverse queries that broaden evidence coverage when the current answer is incorrect, and it discourages redundant search calls when the agent has already arrived at a correct answer. Note that when the answer is correct, R_{query} is non-positive by design; under GRPO's group-relative normalization, only the *relative ranking* among trajectories matters, so a

trajectory with fewer redundant searches receives a higher advantage estimate than one with excessive calls.

3.4 Dense Rewards

Apart from the query and memory rewards, the dense reward includes two additional components: an outcome reward that measures answer correctness via token-level F1, $R_{\text{ans}} = 2|a \cap \hat{a}|/(|a| + |\hat{a}|)$, and a format reward $R_{\text{format}} = -N_{\text{rules}}$ that penalizes structural violations (exceeding T_{max} search turns, invalid actions, or misordered special tokens). A key design principle distinguishes our approach from prior dense reward formulations: our four terms are grounded in the optimization objective of Eq. 2, where R_{ans}, R_{mem}, and R_{query} serve as rule-based proxies for the intractable likelihood terms $\log \pi_\theta(\hat{a})$, $\sum_t \log \pi_\theta(\hat{m}_t)$, and $\sum_t \log \pi_\theta(\hat{q}_t)$, respectively. This contrasts with GIGPO (Feng et al., 2025), which assigns a single discounted return per trajectory and relies on an anchor-state grouping mechanism for credit assignment—an approach that does not differentiate between the functionally distinct roles of query formulation, evidence retention, and answer generation. Because ground-truth trajectories $\hat{m}_t$ and $\hat{q}_t$ are unavailable, the dense reward provides rule-based proxy signals that approximate these terms. All four components are computed via lightweight string operations—token-level F1 for R_{ans}, string containment for R_{mem}, character-level sequence matching for R_{query}, and pattern checking for R_{format}—requiring no auxiliary model inference and adding negligible overhead. The overall reward is

$$R_{\text{Dense}} = R_{\text{ans}} + \alpha \cdot R_{\text{mem}} + \gamma \cdot \mu \cdot R_{\text{query}} + \gamma \cdot R_{\text{format}} \quad (5)$$

where α and γ balance the terms. To stabilize training, we decay the query reward weight μ via a cosine schedule: $\mu = 0.5(1 + \cos(\pi t_{\text{iter}}/t_{\text{decay}}))$ if $t_{\text{iter}} \leq t_{\text{decay}}$ else 0, where t_{iter} is the current training iteration and t_{decay} is the number of decay steps.

3.5 Agentic Reinforcement Learning

To eliminate the need for a separate value estimator, we adopt Group Relative Policy Optimization (GRPO) (Shao et al., 2024) as the RL framework. For a question Q from the dataset D, the policy model π_θ samples an interleaved sequence $y_0, k_0, y_1, k_1, \ldots, y_T, k_T$. Following Search-R1 (Jin et al., 2025), external retrieval tokens k are masked during loss computation, so the objective depends only on the model outputs y.

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_{Q\sim\mathcal{D}, y_i\sim\pi_\theta}\Big[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\min(r_{i,t}\hat{A}_{i,t}, \text{clip}\left(r_{i,t}, 1-\epsilon, 1+\epsilon\right)\hat{A}_{i,t}) - \beta\mathcal{KL}\left[\pi_\theta \,\|\, \pi_{\text{ref}}\right]\Big] \quad (6)$$

where π_θ is the current actor, π_{ref} is a fixed reference policy, and the probability ratio $r_{i,t} = \pi_\theta(y_{i,t} \mid Q, y_{i,<t})/\pi_{\theta_{\text{old}}}(y_{i,t} \mid Q, y_{i,<t})$ is computed against the previous policy. For each question, we sample a group of G outputs and estimate advantages via $\hat{A}_i = (R_i - \text{mean}(R))/\text{std}(R)$, where R is the overall dense reward in Eq. 5.

4 Experiments

4.1 Experiment Settings

Benchmarks and Datasets. We evaluate SE-Search on seven question answering benchmarks that require single-hop or multi-hop retrieval. The single-hop datasets include Natural Questions (NQ) (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), and PopQA (Mallen et al., 2022). The multi-hop datasets include HotpotQA (Yang et al., 2018), 2WikiMultihopQA (Ho et al., 2020), Musique (Trivedi et al., 2022b), and Bamboogle (Press et al., 2022). We use exact match (EM) for all datasets. Following Search-R1, we train SE-Search on a combined training set of NQ and HotpotQA.

Baselines. We compare SE-Search with three classes of methods. (1) Methods without retrieval include direct LLM generation, supervised fine-tuning (SFT), and R1-style training (Guo et al., 2025); (2) Workflows with retrieval include naive RAG that retrieves only from the input question and IRCoT (Trivedi et al., 2022a), which interleaves retrieval with chain-of-thought; (3) Agents with retrieval include retrieval-augmented methods such as Search-o1 (Li et al., 2025c), Search-R1 (Jin et al., 2025), ReSearch (Chen et al., 2025), ZeroSearch (Sun et al., 2025), StepSearch (Wang et al., 2025), EvolveSearch (Zhang et al., 2025a), O^2-Searcher (Mei et al., 2025), InForage (Qian and Liu, 2025), CriticSearch (Zhang et al., 2025b), and AutoRefine (Shi et al., 2025a).

Implementation Details. To emulate a standard search setting, we use the external corpus

Methods	Single-Hop QA				Multi-Hop QA					
	NQ	TriviaQA	PopQA	*Avg.*	HotpotQA	2Wiki	Musique	Bamboogle	*Avg.*	**Avg.**
w/o Retrieval										
Direct Generation	0.106	0.288	0.108	0.167	0.149	0.244	0.020	0.024	0.109	0.134
SFT	0.249	0.292	0.104	0.215	0.186	0.248	0.044	0.112	0.148	0.176
R1-Instruct (Guo et al., 2025)	0.210	0.449	0.171	0.277	0.208	0.275	0.060	0.192	0.184	0.224
R1-Base (Guo et al., 2025)	0.226	0.455	0.173	0.285	0.201	0.268	0.055	0.224	0.187	0.229
Workflow w/ Retrieval										
Naive RAG (Lewis et al., 2020)	0.348	0.544	0.387	0.426	0.255	0.226	0.047	0.080	0.152	0.270
IRCoT (Trivedi et al., 2022a)	0.111	0.312	0.200	0.208	0.164	0.171	0.067	0.240	0.161	0.181
Agent w/ Retrieval										
Search-o1 (Li et al., 2025c)	0.238	0.472	0.262	0.324	0.221	0.218	0.054	0.320	0.203	0.255
Search-R1-Instruct (Jin et al., 2025)	0.397	0.565	0.391	0.451	0.331	0.310	0.124	0.232	0.249	0.336
Search-R1-Base (Jin et al., 2025)	0.421	0.583	0.413	0.472	0.297	0.274	0.066	0.128	0.191	0.312
ReSearch-Instruct (Chen et al., 2025)	0.365	0.571	0.395	0.444	0.351	0.272	0.095	0.266	0.246	0.331
ReSearch-Base (Chen et al., 2025)	0.427	0.597	0.430	0.485	0.305	0.272	0.074	0.128	0.195	0.319
ZeroSearch-Base (Sun et al., 2025)	0.430	0.616	0.414	0.487	0.338	0.346	0.130	0.139	0.238	0.345
StepSearch-Base (Wang et al., 2025)	–	–	–	–	0.329	0.339	0.181	0.328	0.294	–
EXSEARCH-Base (Shi et al., 2025b)	0.368	–	–	–	0.422	0.372	0.138	–	–	–
O^2-Searcher (Mei et al., 2025)	0.444	0.597	0.429	0.490	0.388	0.374	0.160	0.344	0.317	0.391
AutoRefine-Instruct (Shi et al., 2025a)	0.436	0.597	0.447	0.493	0.404	0.380	0.169	0.336	0.322	0.396
AutoRefine-Base (Shi et al., 2025a)	0.467	0.620	0.450	**0.512**	0.405	0.393	0.157	0.344	0.325	0.405
InForage (Qian and Liu, 2025)	0.421	0.597	**0.452**	0.490	0.409	**0.428**	0.172	0.360	0.342	0.405
CriticSearch (Zhang et al., 2025b)	–	–	–	–	0.414	0.409	0.180	0.368	0.343	–
SE-Search-3B (Ours)	**0.475**	**0.624**	0.423	0.507	**0.450**	0.361	**0.183**	**0.424**	**0.355**	**0.420**

Table 2: Accuracy (EM) comparison of SE-Search-3B against baselines using Qwen2.5-3B (Qwen et al., 2025). **Bold** indicates best results. "–" denotes unreported results. Baseline results are from respective papers under the same retriever and corpus.

(Karpukhin et al., 2020) used by Search-R1 and adopt E5-base-v2 (Wang et al., 2022) as the retriever. By default, the retriever returns the top three documents per query, and the backbone LLM is Qwen2.5-3B (Qwen et al., 2025). The query length bounds $L_{\min}$ and $L_{\max}$ are set to 10 and 120 characters. The similarity threshold is $T = 0.3$. The maximum number of search turns is $T_{\max} = 5$, and the decay steps t_{decay} are set to 150. The reward weights are $\alpha = 0.1$ and $\gamma = 0.01$.

4.2 Main Performance

Table 2 compares SE-Search with baseline methods across seven QA benchmarks under identical retriever, corpus, and training data conditions. We highlight two findings.

SE-Search outperforms other methods at the 3B scale. Under the same retriever, corpus, training data, and Qwen2.5-3B setting, SE-Search achieves the highest average EM accuracy among the compared methods, outperforming Search-R1 and recent baselines such as InForage, AutoRefine, and EXSEARCH across the seven benchmarks. The average EM reaches 0.420, representing a 10.8-point absolute improvement and a 34.6% relative gain over Search-R1-Base. Compared with EXSEARCH, which employs iterative self-incentivization with multiple retraining cycles, SE-Search achieves higher accuracy with a single RL training phase.

SE-Search shows particularly strong gains on multi-hop QA benchmarks. The performance improvements are more evident on complex multi-hop QA tasks. For instance, compared with AutoRefine, SE-Search improves HotpotQA by 4.5 percentage points (11.1% relative gain) and Bamboogle by 8 percentage points (23.2% relative gain). Two design choices contribute to these gains: atomic queries decompose the original question into focused subqueries that retrieve more targeted evidence at each step, and memory purification filters out irrelevant content before it can accumulate across reasoning turns. Importantly, SE-Search achieves these gains with greater efficiency—an average of 1.4 search steps per question versus approximately 3.0 for AutoRefine (see Appendix for a detailed efficiency analysis), indicating that the atomic query mechanism retrieves more relevant evidence per step. We note that SE-Search underperforms InForage on 2Wiki (0.361 vs. 0.428), likely because 2Wiki's comparison-style questions favor parallel retrieval over SE-Search's sequential decomposition.

4.3 Ablation Studies

We conduct an ablation study to evaluate the contributions of SE-Search's three key components. For a controlled comparison, all models use the same

Method	Single-Hop QA			Multi-Hop QA				Avg.
	NQ	TriviaQA	PopQA	HotpotQA	2Wiki	Musique	Bamboogle	
Search-R1 (Baseline)	0.424	0.630	0.471	0.346	0.296	0.070	0.169	0.344
+ Memory Purification (MP)	0.426	0.610	0.436	0.377	0.315	0.120	0.260	0.363
Δ vs. Baseline	+0.002	−0.020	−0.035	+0.031	+0.019	+0.050	+0.091	+0.019
+ MP + Atomic Query (AQ)	**0.442**	0.609	0.427	0.381	0.365	0.163	**0.371**	0.394
Δ vs. + MP	+0.016	−0.001	−0.009	+0.004	+0.050	+0.043	+0.111	+0.031
+ MP + AQ + Dense Reward	0.429	**0.661**	**0.467**	**0.436**	**0.374**	**0.177**	0.328	**0.410**
(**SE-Search**, Δ vs. + MP + AQ)	−0.013	+0.052	+0.040	+0.055	+0.009	+0.014	−0.043	+0.016

Table 3: Ablation study on SE-Search components. All models use the same settings, trained for 200 steps, evaluated on 500 samples per benchmark. Each Δ row shows the incremental gain of the newly added component.

retriever and corpus, are trained for 200 steps (62% of full training; component ranking is consistent at 320 steps), and are evaluated on 500 randomly selected samples from each benchmark. Table 3 reports accuracy on seven benchmarks and also reflects how each component changes search behavior through the final performance. We compare four configurations. (1) Search-R1 (baseline); (2) Search-R1 with MP, augmented with Memory Purification and the memory reward R_{mem}; (3) Search-R1 with MP and AQ, adding Atomic Query on top of Memory Purification; (4) Search-R1 with MP, AQ, and DR (SE-Search), the full configuration that further includes Dense Reward R_{Dense};

Each component contributes measurable gains. Memory Purification improves multi-hop benchmarks most substantially—Musique by 5.0 points (71.4% relative) and Bamboogle by 9.1 points (53.9% relative)—by reducing noise in accumulated context. Adding Atomic Query further improves 2Wiki by 15.9%, Musique by 35.8%, and Bamboogle by 42.7% relative, indicating that shorter, non-redundant queries retrieve more informative evidence. The full SE-Search configuration (MP + AQ + DR) achieves the highest average accuracy across all seven benchmarks.

4.4 Scalability

Search agent follows consistent scaling trend. Our main comparison focuses on the 3B scale (Table 2), where SE-Search achieves state-of-the-art results. To examine whether the training recipe transfers to larger backbones, we conduct a scaling study across Qwen2.5-3B, 7B, and 14B while keeping the retriever, corpus, and evaluation protocol unchanged. Table 4 reports EM and F1 scores for each backbone size. The results follow the expected scaling trend (Kaplan et al., 2020): SE-Search-7B achieves an average EM of 0.443 and SE-Search-14B reaches 0.484, with improvements on most individual datasets as model size increases. These results validate the *generalizability* of SE-Search's training recipe, suggesting that the three mechanisms capture general principles of effective search behavior rather than being tailored to a specific model capacity.

Search frequency decreases while accuracy improves. Figure 3 reveals an informative training dynamic: accuracy rises from 0.36 to ∼0.41 (∼14% improvement) while search calls on the training set decrease from 1.53 to 1.32 (∼14% reduction). This inverse relationship suggests that the agent learns to extract more information per retrieval round rather than compensating for low-quality queries with additional searches. Mechanistically, Memory Purification initially drives more search calls as the agent gathers broader evidence for distillation; as training progresses, the Atomic Query reward steers queries toward shorter, more targeted formulations that retrieve relevant documents in fewer rounds. We provide a comprehensive computational efficiency analysis in Appendix A.4, showing that SE-Search achieves 5% faster inference, 12% faster RL updates, 21% shorter responses, and 53% fewer search calls than AutoRefine while maintaining competitive accuracy.

4.5 Efficiency Analysis

A natural concern is whether multi-turn search and dense reward computation introduce prohibitive overhead. We compare SE-Search and AutoRefine under identical hardware; full curves over 250 steps are in Appendix A.4.

Table 5 summarizes the key efficiency metrics. SE-Search achieves **5% faster inference**, **12% faster RL updates**, **21% shorter responses**, and **53% fewer search calls** compared with AutoRefine, while maintaining competitive accuracy (Table 2). The efficiency gains stem from two complementary mechanisms. First, the atomic query reward steers the agent toward shorter, more targeted

Metric	Model	Single-Hop QA			Multi-Hop QA				Avg.
		NQ	TriviaQA	PopQA	HotpotQA	2Wiki	Musique	Bamboogle	
EM	SE-Search-3B	0.475	0.624	0.423	0.450	0.361	0.183	0.424	0.420
	SE-Search-7B	0.477	0.637	0.463	0.453	0.424	0.207	0.440	0.443
	SE-Search-14B	**0.498**	**0.687**	**0.473**	**0.491**	**0.476**	**0.235**	**0.528**	**0.484**
F1	SE-Search-3B	0.567	0.705	0.468	0.573	0.431	0.271	0.536	0.507
	SE-Search-7B	0.567	0.713	0.505	0.571	0.492	0.301	0.575	0.532
	SE-Search-14B	**0.595**	**0.765**	**0.517**	**0.622**	**0.561**	**0.335**	**0.619**	**0.574**

Table 4: Scaling study across Qwen2.5-3B/7B/14B. Both EM and F1 metrics follow a consistent scaling trend.

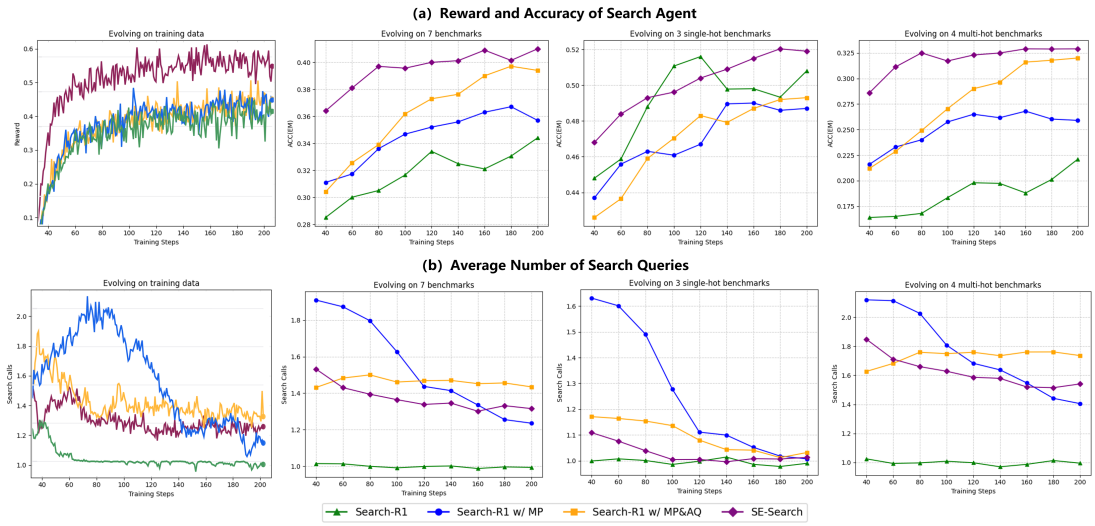

Figure 3: SE-Search's evolution of (a) EM accuracy and (b) search calls on training set and benchmarks.

Metric	SE-Search	AutoRefine
Inference time (500 samples)	580 s	610 s
RL update time (per step)	48 s	55 s
Mean response length (tokens)	950	1,200
Mean search calls (per question)	1.4	3.0
Relative improvement		
Inference speed	+5%	
RL update speed	+12%	
Response conciseness	+21%	
Search call reduction	+53%	

Table 5: Efficiency comparison between SE-Search and AutoRefine. SE-Search achieves competitive accuracy with substantially lower computational cost across all metrics.

queries that retrieve relevant documents in fewer rounds—search calls stabilize at $\sim$1.4 per question, compared with AutoRefine's uncontrolled growth to $\sim$3.0 (Figure 6(f)). Second, the format reward penalizes structural violations and excessive length, producing more concise responses (950 vs. 1,200 tokens). Notably, the four-component dense reward itself is computed via lightweight string operations (token-level F1, string containment, character-level matching, and pattern checking), adding negligible overhead beyond the base RL training loop.

5 Conclusions

We present SE-Search, which co-optimizes memory management, query formulation, and answer generation under one policy π_θ via dedicated dense rewards—unlike prior single-policy agents that rely solely on sparse final-answer signals. Three co-optimized mechanisms drive the agent's improvement: Memory Purification filters noisy retrieved content into a concise memory state via a CEM-based reward; Atomic Query replaces external query rewriters with in-policy learning guided by a substring non-overlap penalty; and a four-component Dense Reward, grounded in the formal optimization objective (Eq. 2), provides functionally distinct feedback on query, memory, answer, and format dimensions through shared GRPO gradients. Experiments on seven QA benchmarks show that SE-Search consistently outperforms recent baselines at the 3B scale, achieving higher accuracy with fewer search steps and lower computational cost than AutoRefine. The efficient training recipe also generalizes to 7B and 14B backbones.

6 Limitations

We discuss several aspects of SE-Search that present opportunities for future improvement.

Static corpus vs. live web search. Following the established evaluation protocol of Search-R1, AutoRefine, and other recent search agent methods, SE-Search retrieves from a fixed Wikipedia dump (Karpukhin et al., 2020) rather than a live search engine. This choice ensures fair comparison and full reproducibility across methods, which would be difficult to guarantee with commercial search APIs due to temporal result variability, API rate limits, and cost. We note that SE-Search's core mechanisms—memory purification, atomic query rewards, and format constraints—are defined over the agent's own generated tokens rather than over retriever-specific features, and are thus in principle compatible with any retrieval backend. Extending SE-Search to live web search is a natural next step that we plan to explore.

Benchmark coverage. Our evaluation spans seven widely used factoid QA benchmarks covering both single-hop and multi-hop reasoning, which is consistent with or broader than most concurrent search agent studies. More demanding tasks that involve deep web browsing (e.g., BrowseComp (Wei et al., 2025)), long-horizon evidence synthesis (e.g., HLE), or cross-lingual retrieval (e.g., X-bench) remain unexplored. These tasks would involve longer trajectories, more diverse document types, and richer tool interactions, offering a valuable testbed for SE-Search's memory and reward mechanisms in future work.

Model scale. Our primary experiments use a 3B backbone, with 7B and 14B results (Table 4) confirming that the training recipe generalizes across these scales. Since the three reward components are computed via lightweight rule-based functions that are agnostic to model size, we expect them to remain applicable at larger scales (e.g., 30B+ or mixture-of-experts architectures), though empirical verification is needed and left for future work.

Hyperparameters and tool scope. The dense reward formulation involves several hyperparameters (α, γ, t_{decay}, similarity threshold T), which we set through a small number of preliminary experiments and kept fixed throughout training. While this manual tuning process was not prohibitively expensive, automated hyperparameter selection could further simplify adoption. Additionally, SE-Search currently supports a single search tool; incorporating complementary capabilities such as page browsing, code execution, or multi-modal retrieval could broaden its applicability to a wider range of knowledge-intensive tasks.

Memory reward granularity. The CEM-based R_{mem} supervises whether the memory contains the *final* answer tokens. For multi-hop questions, intermediate reasoning steps may require bridging entities absent from the final answer string, which CEM cannot directly reward. In practice, the outcome reward R_{ans} still provides indirect supervision for such steps, since retaining useful intermediate evidence ultimately improves the final answer. Designing step-level memory rewards that credit intermediate evidence is a promising direction for future work.

References

Huan ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, and 1 others. A survey of self-evolving agents: On path to artificial super intelligence, 2025. *URL https://arxiv. org/abs/2507.21046*.

Donald O Case and Lisa M Given. 2016. Looking for information: A survey of research on information seeking, needs, and behavior.

Chia-Yuan Chang, Zhimeng Jiang, Vineeth Rakesh, Menghai Pan, Chin-Chia Michael Yeh, Guanchu Wang, Mingzhi Hu, Zhichao Xu, Yan Zheng, Mahashweta Das, and 1 others. 2025. Main-rag: Multi-agent filtering retrieval-augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2607–2622.

Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z Pan, Wen Zhang, Huajun Chen, and 1 others. 2025. Learning to reason with search for llms via reinforcement learning. *arXiv preprint arXiv:2503.19470*.

Yong Deng, Guoqing Wang, Zhenzhe Ying, Xiaofeng Wu, Jinzhen Lin, Wenwen Xiong, Yuqin Dai, Shuo Yang, Zhanwei Zhang, Qiwen Wang, and 1 others. 2025. Atom-searcher: Enhancing agentic deep research via fine-grained atomic thought reward. *arXiv preprint arXiv:2508.12800*.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*.

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).

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*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

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 S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.

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

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

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.

Mei Kobayashi and Koichi Takeda. 2000. Information retrieval on the web. *ACM computing surveys (CSUR)*, 32(2):144–173.

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.

Jian Li, Xiaoxi Li, Yan Zheng, Yizhang Jin, Shuo Wang, Jiafu Wu, Yabiao Wang, Chengjie Wang, and Xiaotong Yuan. 2025a. A survey on ai search with large language models.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, and 1 others. 2025b. Websailor: Navigating super-human reasoning for web agent. *arXiv preprint arXiv:2507.02592*.

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

Xiaoxi Li, Wenxiang Jiao, Jiarui Jin, Guanting Dong, Jiajie Jin, Yinuo Wang, Hao Wang, Yutao Zhu, Ji-Rong Wen, Yuan Lu, and 1 others. 2025d. Deepagent: A general reasoning agent with scalable toolsets. *arXiv preprint arXiv:2510.21618*.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. 2025e. Webthinker: Empowering large reasoning models with deep research capability. *arXiv preprint arXiv:2504.21776*.

Dongqi Liu, Hang Ding, Qiming Feng, Jian Li, Xurong Xie, Zhucun Xue, Chengjie Wang, Jiangning Zhang, and Yabiao Wang. 2026. Disco-rag: Discourse-aware retrieval-augmented generation. *arXiv preprint arXiv:2601.04377*.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting for retrieval-augmented large language models. *Preprint*, arXiv:2305.14283.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. *arXiv preprint arXiv:2212.10511*.

Jianbiao Mei, Tao Hu, Daocheng Fu, Licheng Wen, Xuemeng Yang, Rong Wu, Pinlong Cai, Xinyu Cai, Xing Gao, Yu Yang, and 1 others. 2025. o^2-searcher: A searching-based agent model for open-domain open-ended question answering. *arXiv preprint arXiv:2505.16582*.

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

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*.

Hongjin Qian and Zheng Liu. 2025. Scent of knowledge: Optimizing search-enhanced reasoning with information foraging. *arXiv preprint arXiv:2505.09316*.

Qwen, :, 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, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

Michael Ruse. 1975. Charles darwin's theory of evolution: an analysis. *Journal of the History of Biology*, pages 219–241.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, 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*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.

Yaorui Shi, Shihan Li, Chang Wu, Zhiyuan Liu, Junfeng Fang, Hengxing Cai, An Zhang, and Xiang Wang. 2025a. Search and refine during think: Autonomous retrieval-augmented reasoning of llms. *arXiv e-prints*, pages arXiv–2505.

Zhengliang Shi, Lingyong Yan, Dawei Yin, Suzan Verberne, Maarten de Rijke, and Zhaochun Ren. 2025b. Iterative self-incentivization empowers large language models as agentic searchers. *arXiv preprint arXiv:2505.20128*.

Zhengliang Shi, Lingyong Yan, Dawei Yin, Suzan Verberne, Maarten de Rijke, and Zhaochun Ren. 2025c. Iterative self-incentivization empowers large language models as agentic searchers. *arXiv preprint arXiv:2505.20128*.

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. 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.

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

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

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

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Gang Fu, Yong Jiang, and 1 others. 2025. Webdancer: Towards autonomous information seeking agency. *arXiv preprint arXiv:2505.22648*.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Kristian Kersting, Jeff Z Pan, Hinrich Schütze, and 1 others. 2025. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, and 1 others. 2025. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*.

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

Ding-Chu Zhang, Yida Zhao, Jialong Wu, Liwen Zhang, Baixuan Li, Wenbiao Yin, Yong Jiang, Yu-Feng Li, Kewei Tu, Pengjun Xie, and 1 others. 2025a. Evolvesearch: An iterative self-evolving search agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 13134–13147.

Yaocheng Zhang, Haohuan Huang, Zijun Song, Yuanheng Zhu, Qichao Zhang, Zijie Zhao, and Dongbin Zhao. 2025b. Criticsearch: Fine-grained credit assignment for search agents via a retrospective critic. *arXiv preprint arXiv:2511.12159*.

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).

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, and 1 others. 2025. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*.

A Appendix

A.1 Training and Evaluation Datasets

Following Search-R1 and other RL-based methods, we merged NQ and HotpotQA to form the training dataset, comprising 169,615 samples. The evaluation set comprises 51,713 samples drawn from the test set of four benchmarks (NQ, TriviaQA, PopQA, Bamboogle) and the development sets of three benchmarks (HotpotQA, 2Wiki, and Musique).

A.2 GRPO Hyperparameters

We list the key hyperparameters used with the VeRL framework (Sheng et al., 2025). In Table 6, we adopt the same data, actor, and rollout configuration as Search-R1 and AutoRefine. For the dense reward design, the atomic query reward is computed for query lengths in the range from 10 to 120 and uses a similarity threshold of 0.3. The decay step is set to 150, which is half of the total training steps.

Module	Hyper-parameter	Value
Data	Max Documents Length	512
	Total Training Steps	320
	Max Response Length	2048
	Retriever Top-k	3
Actor	Training Batch Size	256
	Micro Training Batch Size	64
	Learning Rate	1×10^{-6}
	KL Coefficient β	0.001
	Clip Ratio ϵ	0.2
Rollout	Max Search Actions	5
	Group Size G	10
	Temperature	1.0
	Top-p	0.95
Reward	$L_{\min}$	10
	$L_{\max}$	120
	Similarity Threshold T	0.3
	t_{decay}	150
	Reward weight α	0.1
	Reward weight γ	0.01

Table 6: GRPO hyperparameters for RL training.

A.3 Additional Experiment Results

SE-Search calls the search engine more often on complex questions. Figure 3(b) compares search frequency across training steps and question types, where multi-hop questions represent higher compositional complexity than single-hop ones. Compared with Search-R1, which relies on a fixed sin-

gle search call, SE-Search issues an average of approximately 1.54 search calls on multi-hop questions while remaining close to 1.0 on single-hop benchmarks. This pattern suggests that SE-Search adjusts tool use to question complexity, allocating more retrieval to gather evidence for multi-hop reasoning while avoiding unnecessary calls on simpler questions.

SE-Search produces more atomic and diverse queries. Prior work emphasizes higher search frequency while largely overlooking query quality. Figure 4(a) reports the average character length and variance of search queries for three methods, including SE-Search at approximately 50, AutoRefine at approximately 65, and Search-R1 at approximately 90. These results suggest that shorter and more focused queries are more effective at retrieving relevant and high-quality documents. Figure 4(b) further compares query diversity using the similarity ratio defined in Eq. 3. SE-Search achieves a similarity ratio of about 0.5, whereas AutoRefine reaches a higher value of about 0.6. Higher query similarity indicates greater redundancy in retrieved results, which may introduce repetitive or biased information and degrade the contextual quality provided to the LLM.

Memory Purification stores key information for answers. We evaluate the information stored in memory using CEM, which measures whether the memory contains evidence related to the ground-truth answer. Figure 4(c) shows that the mean CEM of the memory reaches approximately 0.65 at the final stage of training. This result demonstrates that the agent successfully extracts useful information from retrieved documents and stores key facts in memory, rather than retaining noisy or irrelevant details. Consistent with the training objective, the blue line in Figure 3(b) indicates that the agent issues more search actions during early training to purify memory with important information and filter noise, which leads to higher memory rewards and more reliable memory contents.

Dense Rewards help convergence. We further examine the effect of dense rewards, which introduce a format reward and replace the EM-based outcome reward with an F1-based reward. Figure 4(c) shows that dense rewards improve the accuracy of memory content during the early stages of training, indicating earlier learning of how to record answer-relevant evidence. In particular, the format reward discourages invalid outputs and reduces unproductive exploration trajectories, thereby accelerating convergence with fewer RL exploration steps. Moreover, the F1-based outcome reward provides graded feedback even when the answer is not an exact match, which yields denser learning signals than EM and stabilizes training. Consistent with this, Figure 3(a) shows that SE-Search with dense rewards attains higher reward values than SE-Search without dense rewards, and this improvement is mainly attributed to the F1-based outcome reward.

Per-Benchmark Training Dynamics. To provide a more comprehensive characterization of SE-Search, Figure 5 shows its training dynamics, plotted as average accuracy, across all seven benchmarks: Natural Questions (NQ), TriviaQA, PopQA, Bamboogle, HotpotQA, 2Wiki, and Musique. The curves indicate that most benchmarks exhibit steady improvement throughout training, with multi-hop tasks showing more pronounced gains in later stages as the agent refines its search and memory strategies.

A.4 Computational Efficiency Analysis

We comprehensively evaluate the training and inference efficiency of SE-Search to address the natural concern that multi-turn search and dense reward computation may introduce prohibitive overhead. Figure 6 presents six key metrics tracked across 250 training steps, comparing four methods: Search-R1 (baseline), Search-R1 w/ MP (with Memory Purification), Search-R1 w/ MP&AQ (our full SE-Search), and AutoRefine.

Training Efficiency Remains Competitive As shown in Figure 6(a), SE-Search achieves similar throughput to AutoRefine ($\sim$1,100 tokens/s), representing a moderate 15% reduction compared to Search-R1's peak (1,300 tokens/s). Critically, our update actor time (Figure 6(d)) stabilizes at approximately 48s per step, significantly lower than AutoRefine's 55s and with less variance. This demonstrates that our four-component dense reward design does not impose excessive computational burden during RL optimization.

Inference Cost-Performance Trade-off While multi-turn search increases inference time, SE-Search maintains an advantage over AutoRefine. Figure 6(c) shows that SE-Search requires 580s for 500 samples testing compared to AutoRefine's

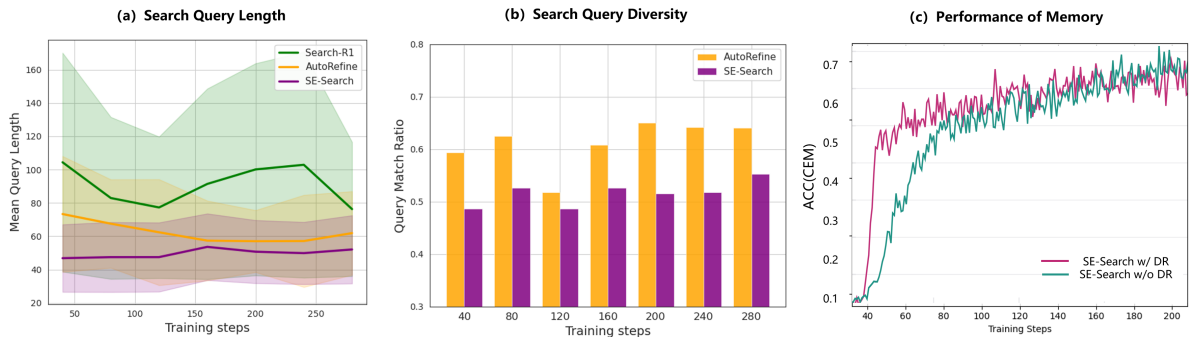

Figure 4: Statistics for (a) search query lengths, (b) search query diversity, (c) accuracy of memory contents.

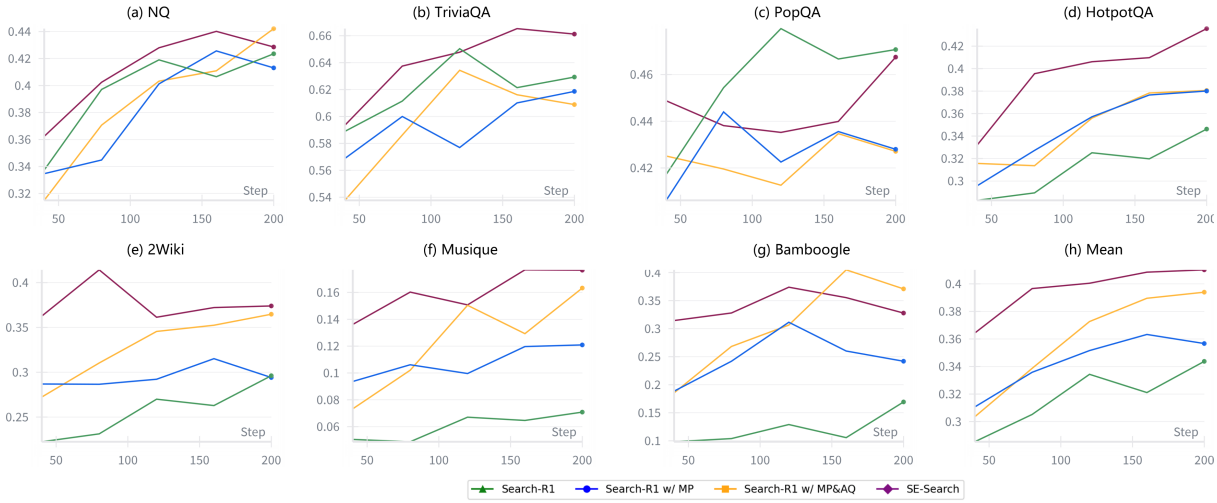

Figure 5: SE-Search's evolution of EM accuracy across seven benchmarks during training.

610s, despite achieving comparable accuracy. The per-token generation time (Figure 6(b)) converges to similar levels across methods (∼0.07ms), indicating efficient implementation. Notably, SE-Search generates more concise responses (950 tokens in Figure 6(e)) than AutoRefine's verbose outputs (1,200 tokens), attributable to our format reward that penalizes excessive length.

Learned Search Efficiency The most striking finding emerges from Figure 6(f): AutoRefine exhibits uncontrolled search frequency growth, escalating from 2.6 to 3.0 valid searches per question by step 200. In contrast, SE-Search stabilizes at approximately 1.4 searches through our atomic query mechanism. This learned efficiency demonstrates that SE-Search adapts search behavior to question complexity—allocating more retrievals to multi-hop reasoning while avoiding redundant searches on simple queries—whereas AutoRefine's unconstrained exploration incurs unnecessary computational costs without proportional accuracy gains.

In summary, SE-Search achieves superior cost-efficiency compared to AutoRefine: 5% faster inference, 12% faster RL updates, 21% shorter responses, and 53% fewer search calls, while maintaining competitive accuracy.

A.5 Hyperparameter Sensitivity Analysis

We analyze the sensitivity of SE-Search to three key hyperparameters in the atomic query reward: the similarity threshold T, the minimum query length $L_{\min}$, and the maximum query length $L_{\max}$. For each analysis, we vary one parameter while keeping all others at their default values (T=0.3, $L_{\min}$=10, $L_{\max}$=120). Table 7 reports the average EM accuracy across the seven evaluation benchmarks.

The results show that SE-Search is robust to the choice of these hyperparameters. Across all settings, the performance fluctuation remains within 1.5 percentage points of the default configuration. The similarity threshold T is the most sensitive among the three: values from 0.2 to 0.4 all yield comparable accuracy (within 0.4 points), while more extreme values (T=0.1 or T=0.5) cause a modest drop, indicating that the substring

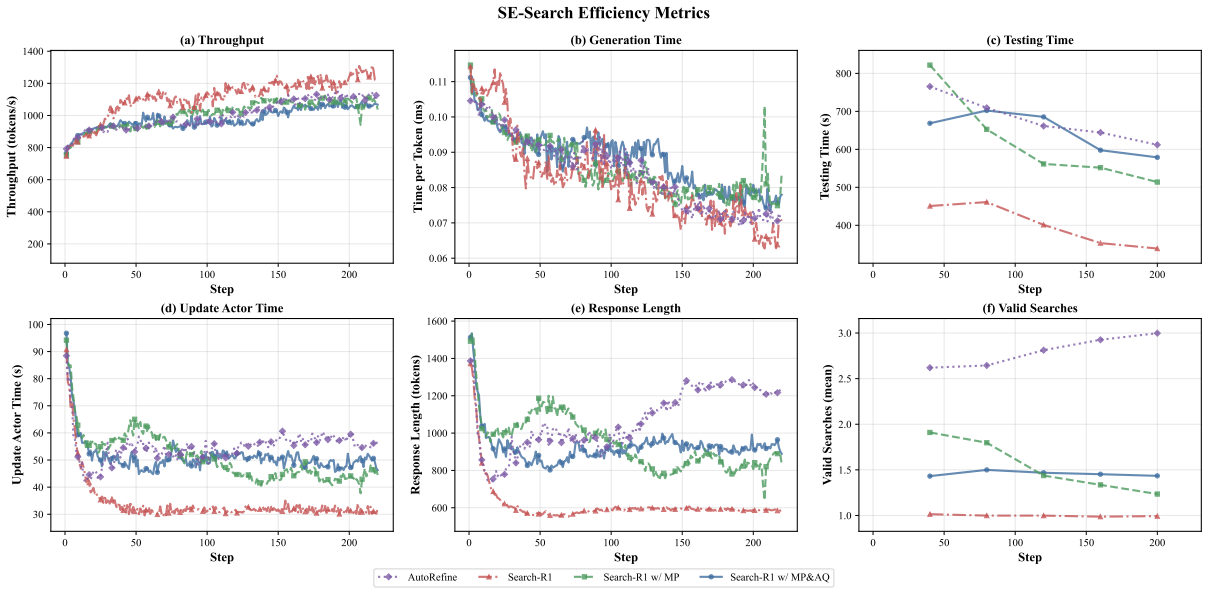

Figure 6: Computational efficiency analysis of SE-Search across 250 training steps. We compare Search-R1 (baseline, red), Search-R1+MP (with Memory Purification, green), Search-R1+MP+AQ (full SE-Search, blue) and AutoRefine (Shi et al., 2025a) across six metrics: (a) Training throughput in tokens per second, (b) Generation time per token, (c) Total testing time on evaluation set, (d) Update actor time per RL step, (e) Mean response length in tokens, and (f) Mean number of valid atomic queries per question. SE-Search maintains competitive training efficiency while learning to adaptively reduce search frequency from 1.5 to 1.35 calls (f), partially offsetting inference costs.

Param	Values	Avg EM (%)
T	0.1	40.5
	0.2	41.6
	0.3 (default)	**42.0**
	0.4	41.7
	0.5	41.2
$L_{\min}$	5	41.5
	10 (default)	**42.0**
	15	41.8
	20	41.4
$L_{\max}$	80	41.3
	100	41.7
	120 (default)	**42.0**
	150	41.9

Table 7: Hyperparameter sensitivity analysis. Each row varies one parameter while keeping others at default. Performance remains stable across a wide range of values, with less than 1.5 percentage-point absolute variation in all cases.

non-overlap penalty functions effectively across a broad tolerance range. Similarly, the query length bounds $L_{\min}$ and $L_{\max}$ exhibit graceful degradation at boundary values—overly restrictive bounds ($L_{\min}$=20 or $L_{\max}$=80) slightly reduce performance by constraining query expressiveness, while overly permissive bounds have minimal negative effect. This stability suggests that practitioners can adopt the default settings without extensive tuning.

A.6 Related Works

Retrieval-Augmented Generation. Retrieval-Augmented Generation (RAG) grounds LLM outputs in external knowledge (Lewis et al., 2020). A typical RAG system couples a retriever with a generator; surveys identify four common paradigms (Gao et al., 2023): Sequential, Branching, Conditional, and Loop RAG. Most conventional systems follow a fixed retrieve-then-generate workflow and do not permit the model to decide *when* or *what* to search during reasoning. Recent efforts have also explored improving the quality of retrieved context itself; for example, Disco-RAG (Liu et al., 2026) leverages discourse structure to select more coherent and contextually grounded passages, which complements approaches that focus on the generation side. SE-Search departs from the standard RAG paradigm along a different axis by targeting autonomous, multi-turn search behavior with explicit memory management, query exploration, and behavioral constraints.

Search Agent. Search agents extend RAG by invoking search tools within multi-step reasoning (Li et al., 2025c,e; Wu et al., 2025). Training-oriented methods adopt different strategies: Search-R1 (Jin et al., 2025) and ReSearch (Chen et al., 2025) use single-phase RL with sparse final-answer rewards; EXSEARCH (Shi et al., 2025c) and EvolveSearch (Zhang et al., 2025a) employ iterative data-generation–retraining cycles; AutoRefine (Shi et al., 2025a) and WebSailor (Li et al., 2025b) introduce context refinement steps. On the memory side, Memory-R1 (Yan et al., 2025) separates memory and answer generation into two distinct agents that communicate through a shared state, and AgentFold (Ye et al., 2025) manages long-horizon context via proactive compression using an auxiliary module. SE-Search differs from these approaches by co-optimizing memory management, query formulation, and answer generation within a single policy through structured dense rewards—providing targeted feedback on query, memory, answer, and format dimensions simultaneously, rather than relying on sparse final-answer signals as in Search-R1 or inter-agent communication as in Memory-R1. The Think-Search-Memorize loop and Atomic Query mechanism address the common limitation that retrieved noise compounds across reasoning steps.

Reinforcement Learning. Reinforcement learning (RL) provides a general framework for improving LLM behavior. Common approaches include proximal policy optimization (PPO) (Schulman et al., 2017), direct preference optimization (DPO) (Rafailov et al., 2023), and group relative policy optimization (GRPO) (Shao et al., 2024). Extensions such as dynamic sampling policy optimization (DAPO) (Yu et al., 2025) and group sequence policy optimization (GSPO) (Zheng et al., 2025) further improve training stability and efficiency. GRPO uses group-based normalization and avoids a separate critic. For multi-turn agent training, GIGPO (Feng et al., 2025) introduces step-level credit assignment via anchor state grouping but still produces a single scalar signal per trajectory. SE-Search takes a complementary approach: rather than refining credit assignment within a single reward, it decomposes the reward into four functionally distinct components—answer quality, memory content, query formulation, and format compliance—each grounded in a specific term of the optimization objective and providing targeted feedback on a different dimension of search agent behavior.

A.7 Case Studies

Table 8 lists the atomic queries derived from a complex user question, illustrating the self-planning and self-exploration of SE-Search. Table 9 and Table 10 show the memory contents and full trajectories generated by SE-Search for two randomly selected examples from HotpotQA and Musique. These results demonstrate SE-Search's self-memory and self-discipline.

[1]Ground-truth answers are taken verbatim from the 2WikiMultihopQA benchmark annotation.

Question: Which film has the director died earlier, Kati Patang or A Thief In The Dark?
Ground Truth: A Thief In The Dark

SE-Search Trajectory

Step 1. <search> Director of Kati Patang </search>
<memory> ... the director of Kati Patang is Shakti Samanta. </memory>

Step 2. <search> Director of A Thief In The Dark </search>
<memory> ... the director of A Thief In The Dark is Donald W. Thompson. </memory>

<think> I need to determine which director died earlier, Shakti Samanta or Donald W. Thompson. </think>

Step 3. <search> Death date of Shakti Samanta </search>
<memory> ... Shakti Samanta died on 9 April 2009. </memory>

Step 4. <search> Death date of Donald W. Thompson </search>
<memory> ... Thompson died on February 19, 2011. </memory>

<answer> A Thief In The Dark </answer>

Table 8: Atomic query decomposition example illustrating SE-Search's self-planning and self-exploration capabilities.[1]

Question: Who is the spouse of the actor who played Scarlett in Gone with the Wind?
Ground Truth: Laurence Olivier

SE-Search Trajectory

To answer this question, I'll follow these steps.

Step 1. <think> I need to identify the actor who played Scarlett in "Gone with the Wind". </think>

Step 2. <search> Actor who played Scarlett in Gone with the Wind </search>
<documents> Doc 1 (Title: "Gone with the Wind (film)") **The leading roles are played by Vivien Leigh (Scarlett), Clark Gable (Rhett), Leslie Howard (Ashley), and Olivia de Havilland (Melanie).** ... Doc 3 (Title: "Vivien Leigh") **She won two Academy Awards for Best Actress, for her iconic performances as Scarlett O'Hara in "Gone with the Wind" (1939)** </documents>

Step 3. <memory> The actor who played Scarlett in "Gone with the Wind" is Vivien Leigh. </memory>

Step 4. <think> Now I need to find who Vivien Leigh is married to. </think>

Step 5. <search> Vivien Leigh husband </search>
<documents> Doc 1 (Title: "Vivien Leigh") ..., **the public strongly identified Leigh with her second husband, Laurence Olivier**, who was her spouse from 1940 to 1960. ... </documents>

Step 6. <memory> Vivien Leigh is married to Laurence Olivier. </memory>

Step 7. <answer> Laurence Olivier </answer>

Table 9: Case study (HotpotQA): SE-Search demonstrates self-memory and self-discipline. Key evidence is highlighted in **bold dark blue**.

Question: How many ethnic minorities were looked at differently in the city where the headquarters of the only group larger than Vilaiyaadu Mankatha's record label is located?

Ground Truth: two

SE-Search Trajectory

To answer this question, let's follow these steps.

Step 1. <think> First, I need to find out which group larger than Vilaiyaadu Mankatha's record label is located in the headquarters city. </think>

Step 2. <search> Which group larger than Vilaiyaadu Mankatha's record label is located in the headquarters city? </search>

<documents> Doc 1 (Title: "Tharangini Records") Primarily established as the first recording and distributing music label in Kerala for Malayalam films... **Tharangini Records (currently Tharangni) is the music company** founded by noted Indian Playback singer K. J. Yesudas in 1980 at Thiruvananthapuram, Kerala. </documents>

Step 3. <memory> ..., the group larger than Vilaiyaadu Mankatha's record label is Tharangini Records. </memory>

Step 4. <search> What city is the headquarters of Tharangini Records? </search>

<documents> Doc 1 (Title: "Tharangini Records") ... **The office and studio were moved to Abhiramapuram, Chennai, Tamil Nadu licensed under Dr. K. J. Yesudas Audio and Visual Corporation Ltd.** ... </documents>

Step 5. <memory> ..., the headquarters of Tharangini Records is located in Chennai, Tamil Nadu. </memory>

Step 6. <search> How many ethnic minorities were looked at differently in the city of Chennai, Tamil Nadu? </search>

<documents> Doc 1 (Title: "Demographics of Chennai") ... however, the population growth rate was slow. Hindus form the majority of Chennai's population **but the city also has substantial Muslim and Christian minorities.** </documents>

Step 7. <memory> ..., the city of Chennai, Tamil Nadu has a substantial Muslim and Christian minority. </memory>

Step 8. <answer> Two </answer>

Table 10: Case study (Musique): SE-Search demonstrates multi-hop reasoning with progressive memory refinement. Key evidence is highlighted in **bold dark blue**.