

Fake It Till You Make It: Training Deep-Search Agents in Synthetic Search Environments

Yibo Xue^{1*} Pin Zheng^{1*} Chengyu Wang^{2†} Yukun Kang¹ Yuanjie Lyu²
Ming Wang² Jian Peng¹ Jun Huang² Changliang Xu^{1†}

¹Hangzhou Institute for Advanced Study, UCAS ²Alibaba Cloud Computing

{xueyibo24, zhengpin24, kangyukun23}@mailsucas.ac.cn

{pengj, xuchangliang}@ucas.ac.cn

{chengyu.wcy, lyuyuanjie.lyj, jinpu.wm, huangjun.hj}@alibaba-inc.com

Abstract

Deep-search agents are commonly trained via repeated interaction with real-world search APIs and browser environments. While realistic, this paradigm is expensive, difficult to reproduce, and sensitive to external drift. Existing simulator-based alternatives reduce dependence on live retrieval, but they largely inherit task distributions from pre-existing QA datasets, limiting their alignment with the behaviors required for deep search. We introduce SYNSEARCH, a framework for training deep-search agents entirely within a synthetic search-browse environment. Our core idea is to jointly design the environment and the training-task distribution, enabling the synthesis of *environment-aligned* tasks and reinforcement learning of compact language models in multi-turn search settings. By grounding task validity in executable environment trajectories rather than surface-level plausibility alone, SYNSEARCH unifies task generation, trajectory verification, and policy optimization within a low-marginal-cost and reproducible framework. Experiments on multi-hop QA and deep-search benchmarks show that SYNSEARCH consistently improves both answer accuracy and search behavior, with especially large gains on deep-search tasks. Moreover, SYNSEARCH offers low marginal cost, reproducibility, controllability, and strong performance under real-search evaluation, highlighting synthetic environments as an effective substrate for training deep-search agents.

1 Introduction

Large language models (LLMs) are increasingly expected to act as *information-seeking agents* rather than passive text generators (Jin et al., 2025). In realistic settings, answering a complex question

often requires more than a single retrieval step: an agent must issue targeted queries, inspect retrieved pages, accumulate evidence across interactions, and revise its strategy as new information emerges (Huang et al., 2025). We refer to this capability as *deep search*. Unlike standard retrieval-augmented generation, deep search is inherently a sequential decision-making problem that requires effective multi-turn interaction with search and browsing tools.

Recent work suggests that reinforcement learning can improve such behavior in tool-augmented environments (Jin et al., 2025; Song et al., 2025; Chen et al., 2025b; Wei et al., 2025b; Zheng et al., 2025). However, training deep-search agents in real search environments remains expensive and difficult to control: it requires repeated calls to external APIs, retrieved content can drift over time, and browser interaction introduces substantial engineering overhead (Sun et al., 2025a; Fan et al., 2025). As a result, long-horizon search training is costly, hard to reproduce, and ill-suited to controlled analysis.

A natural alternative is to replace live search with simulation. For deep search, however, simulating the retrieval backend alone is not sufficient. Recent approaches such as ZeroSearch (Sun et al., 2025a) and SSRL (Fan et al., 2025) reduce reliance on live search APIs, but they largely inherit task distributions from existing QA datasets. This creates a mismatch between the tasks seen during training and the behaviors required at test time: if training tasks do not require multi-step retrieval, browse-dependent evidence gathering, or adaptive query reformulation, then a cheaper environment alone is unlikely to produce a strong deep-search agent. In this setting, the training-task distribution is as important as the retrieval backend itself.

Our central premise is that the environment

*Equal contribution.

†Corresponding authors.

and the training tasks should be designed jointly. Tasks should be constructed under the same search–browse interface used for policy optimization and retained only if they induce verifiable interaction trajectories in that environment. This distinction matters because search snippets rarely provide sufficient evidence on their own; strong performance often depends on selective browsing, cross-page evidence aggregation, and iterative query refinement.

We introduce SYNSEARCH, a framework for training deep-search agents entirely within a synthetic search–browse environment. SYNSEARCH synthesizes *environment-aligned* tasks directly in the target environment through a closed-loop multi-agent pipeline with iterative refinement, rollback-based correction, difficulty control, filtering, and solver-in-the-loop verification. It then trains compact language models in the same environment using multi-turn reinforcement learning with process-aware rewards. We formalize deep search as a task-conditioned partially observable Markov decision process (POMDP) with structured tool-use actions over search and browsing interfaces, and define task validity through executable trajectories rather than surface plausibility alone. The result is a unified and low-marginal-cost framework for task synthesis, trajectory validation, and policy optimization.

Experiments across Qwen2.5-3B/7B, Llama3.1-8B, and Qwen3-8B support this perspective. Compared with training on existing QA datasets or prior synthetic data, SYNSEARCH consistently improves both answer quality and search behavior, yielding longer trajectories, more browsing actions, and stronger downstream performance on both multi-hop QA and browse-intensive deep-search benchmarks. Process-aware rewards further reduce redundant querying and encourage more deliberate search behavior.

In summary, our contributions are as follows:

1. We identify a key limitation of prior simulated deep-search training: simulating the retrieval backend without jointly designing the training-task distribution is insufficient to induce long-horizon search–browse behavior.
2. We propose SYNSEARCH, a framework for training deep-search agents entirely within a synthetic search–browse environment by coupling environment-aligned task synthesis, trajectory validation, and reinforcement learning in a shared reproducible setting.
3. We formalize deep search as a task-conditioned POMDP with structured tool-use actions, and define task validity through executable trajectories in the environment rather than surface plausibility alone.
4. We show that environment-aligned tasks and process-aware rewards improve both answer accuracy and learned search behavior across multiple model families and benchmark settings.

2 Related Work

Search RL and deep-search agents. A growing body of work trains LLMs to perform search through iterative tool use and reinforcement learning (Jin et al., 2025; Song et al., 2025; Chen et al., 2025b; Li et al., 2025b; Tan et al., 2026). Extensions to web-agent settings incorporate online browsing and long-horizon exploration (Wei et al., 2025b; Zheng et al., 2025; Li et al., 2025c; Nakano et al., 2021; Yao et al., 2022), while recent deep-search systems such as ASearcher and WebDancer place greater emphasis on rich search behavior (Gao et al., 2025; Wu et al., 2025a). These studies demonstrate that search competence can be learned, but they typically rely on costly live environments or assume fixed task distributions.

Simulated search environments. To reduce the cost and instability of live search, another line of work replaces external retrieval with simulated search or internal knowledge access. ZeroSearch uses an LLM-based retrieval simulator during reinforcement learning (Sun et al., 2025a), while SSRL studies self-search by treating the model’s own parametric knowledge as a searchable environment (Fan et al., 2025). Related work on retrieval- and search-augmented generation explores pseudo-documents, retrieval surrogates, and adaptive retrieval control (Yu et al., 2022; Asai et al., 2024; Borgeaud et al., 2021; Jiang et al., 2023; Trivedi et al., 2022; Shi et al., 2023; Gao et al., 2022; Guu et al., 2020; Izacard et al., 2022). In contrast to our setting, these approaches primarily modify the retrieval backend rather than jointly designing the training-task distribution and the environment.

Synthetic task generation for agents. Synthetic data has been widely used to train reasoning, tool-use, and information-seeking agents (Wang et al., 2023; Xu et al., 2025; Liu et al., 2023; Qin et al., 2023; Mialon et al., 2023; Shi et al., 2025). Closest to our setting, ASearcher uses synthetic QA construction for long-horizon search training, while

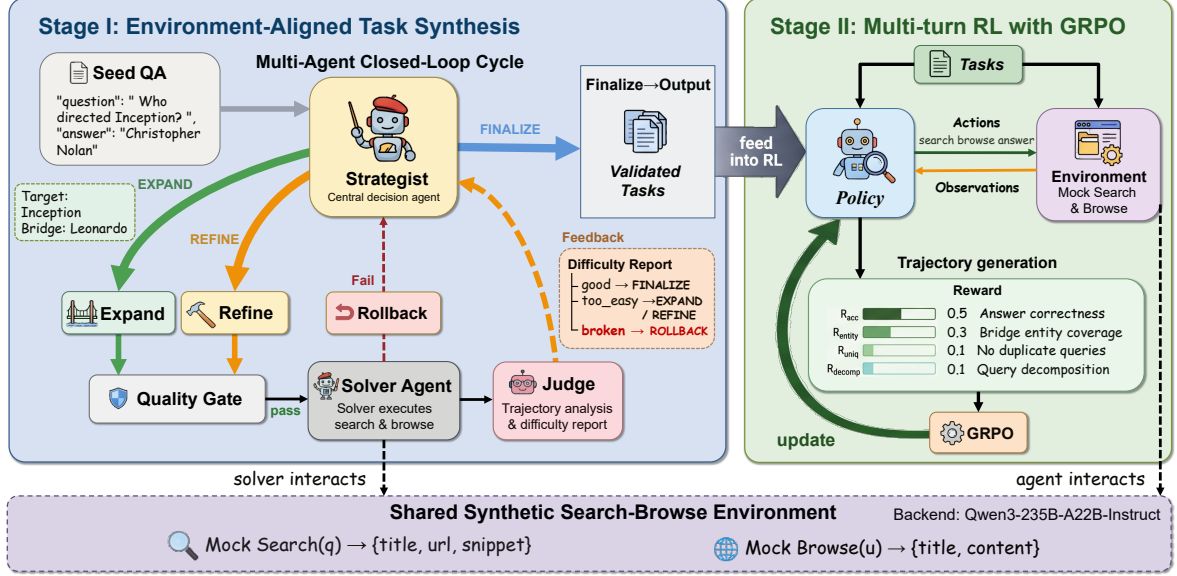


Figure 1: **Overview of SYNSEARCH.** SYNSEARCH unifies environment-aligned task synthesis, solver-in-the-loop trajectory validation, and multi-turn reinforcement learning within a shared synthetic search–browse environment. This shared-interface design enables task construction, trajectory verification, and policy optimization to reinforce one another in a single low-marginal-cost and reproducible training framework.

WebShaper, WebSailor, and related work aim to create information-seeking tasks with richer reasoning or retrieval structure (Gao et al., 2025; Tao et al., 2025; Li et al., 2025a; Sun et al., 2025b). A parallel line of work builds compact agentic models for general tool use, either by pairing synthetic task construction with iterative data flywheels (Lyu et al., 2026b), or by training on synthetic tasks inside simulated tool environments under rubric-based rewards (Lyu et al., 2026a). Our work differs in that task synthesis is performed directly in the same synthetic search–browse environment used for policy learning, and tasks are retained only when solver-in-the-loop interaction verifies that they admit executable and informative trajectories.

Position of our work. Our work lies at the intersection of these directions, but differs in its core design objective. We treat the *training-task distribution* as a first-class design variable and couple task synthesis, trajectory validation, and policy learning within a shared synthetic search–browse environment. Unlike ASearcher and WebShaper, which decouple synthetic task construction from training in real search environments, SYNSEARCH couples task generation, behavioral validation, and policy optimization within a single environment $\mathcal{E}_{\text{syn}}$.

3 SYNSEARCH: Proposed Framework

Figure 1 presents an overview of SYNSEARCH. We formalize deep search as a task-conditioned sequential decision-making problem and instantiate it in a shared synthetic search–browse environment. SYNSEARCH consists of three tightly coupled components: environment-aligned task synthesis, solver-in-the-loop trajectory validation, and multi-turn reinforcement learning. Candidate tasks are constructed and filtered in the same environment later used for policy optimization, ensuring that the training distribution induces the search behaviors the policy is expected to learn.

3.1 Problem Formulation

Deep search as a task-conditioned POMDP. Let $\mathcal{X}$ denote the task space, and let p be a training distribution over $\mathcal{X}$. For each task $x \in \mathcal{X}$, the agent interacts with an episode-specific environment $\mathcal{E}_x$. We model this interaction as a task-conditioned partially observable Markov decision process (POMDP). At step t , the agent receives an observation o_t , maintains a history $h_t = (x, o_1, a_1, \dots, o_t)$, and selects an action according to a policy $\pi_\theta(a_t | h_t)$ over a horizon H . The action space consists of structured tool-use actions $\mathcal{A} = \{\text{search}(q), \text{browse}(u), \text{answer}(y)\}$, where q is a query, u is a URL, and y is a final answer. The action $\text{answer}(y)$ terminates

the episode, and a trajectory is written as $\tau = (x, o_1, a_1, \dots, o_H, a_H)$. Given rewards $\{r_t\}_{t=1}^H$, the generic objective is

$$J(\theta; p) = \mathbb{E}_{x \sim p} \left[\mathbb{E}_{\tau \sim p(\tau|x; \pi_\theta, \mathcal{E}_x)} \left[\sum_{t=1}^H \gamma^{t-1} r_t \right] \right]. \quad (1)$$

Equation (1) highlights that learning depends not only on the policy and reward function, but also on the training-task distribution, which determines the trajectories encountered during optimization.

Environment alignment and task validity. Our key premise is that task quality should be defined behaviorally within the target environment rather than by surface plausibility alone. Let $\phi(x, \mathcal{E}) \in \mathbb{R}^m$ denote a behavioral descriptor extracted from solving task x in environment $\mathcal{E}$, such as trajectory length, browse frequency, query diversity, or intermediate-entity coverage. The induced behavioral profile of a task distribution p in environment $\mathcal{E}$ is $\Phi(p, \mathcal{E}) = \mathbb{E}_{x \sim p}[\phi(x, \mathcal{E})]$. Given a target profile Φ^* , we say that p is *environment-aligned* in $\mathcal{E}$ if $\Phi(p, \mathcal{E})$ is sufficiently close to Φ^* under a chosen discrepancy measure. We use this notion as a conceptual lens for distribution design rather than as a separately optimized objective.

To assess validity at the task level, we evaluate candidate tasks through repeated solver interaction in the target environment. For a candidate task x , let $\mathcal{T}(x) = \{\tau^{(1)}, \dots, \tau^{(K)}\}$ denote repeated solver trajectories, and let $\psi(\tau) \in \mathbb{R}^d$ denote a trajectory-quality descriptor capturing properties such as answer correctness, total interaction steps, difficulty level, and shortcut risk. We define the empirical trajectory profile of x as $\bar{\psi}(x) = \frac{1}{K} \sum_{k=1}^K \psi(\tau^{(k)})$. Let $\Omega \subseteq \mathbb{R}^d$ be an admissible region encoding solvability, behavioral richness, stability, and low shortcut risk. We regard a task as valid when $\bar{\psi}(x) \in \Omega$. In practice, Ω is instantiated through threshold-based criteria derived from solver trajectories and judge outputs (Appendix C). A task is therefore retained only if it induces executable, stable, and behaviorally informative interaction in the same environment later used for policy learning.

3.2 Shared Synthetic Environment

SYNSEARCH uses a shared synthetic search-browse environment $\mathcal{E}_{\text{syn}}$ for both task synthesis and policy optimization. We instantiate $\mathcal{E}_{\text{syn}}$ with a high-capacity LLM backend (e.g., Qwen3-235B-A22B) to provide broad knowledge coverage while

maintaining a reproducible and low-marginal-cost interface. The goal is to simulate the *tool interface* of search and browsing rather than expose answers directly. Unlike answer-conditioned simulators (Sun et al., 2025a), the tools are prompted only with information naturally available at inference time, which helps reduce answer leakage during both synthesis and training. Implementation details are provided in Appendix D.

Search tool. Given a query q , the search tool returns a ranked result list $\text{Search}(q) = \{(t_i, u_i, s_i)\}_{i=1}^N$, where t_i , u_i , and s_i denote the title, URL, and snippet of the i -th result.

Browse tool. Given a URL u , the browse tool returns page content $\text{Browse}(u; c) = (u, t, d)$, where t is the page title, d is the page content, and c is optional browsing context.

Why model both actions? Search proposes candidate directions, whereas browsing is needed to inspect and aggregate evidence that snippets alone typically do not provide. Search-only simulators may support query reformulation, but they do not train the policy to decide when and how to inspect retrieved pages.

3.3 Environment-Aligned Task Synthesis

SYNSEARCH synthesizes training tasks by interleaving candidate generation, solver-based verification, and feedback-driven revision within $\mathcal{E}_{\text{syn}}$.

Strategist-controlled generation. Starting from a seed question, a Strategist agent iteratively selects a meta-action $m_k \in \{\text{EXPAND}, \text{REFINE}, \text{ROLLBACK}, \text{FINALIZE}\}$.

EXPAND introduces bridge entities or additional reasoning hops; REFINE reduces shortcuts and strengthens distractors while preserving answerability; ROLLBACK returns to a previously valid state; and FINALIZE exports the task once it is sufficiently difficult and behaviorally informative.

Filtering and solver-in-the-loop validation. Each candidate first passes through a quality filter that removes flawed cases, such as answer leakage, broken logic, pseudo-multi-hop structure, or non-unique answers. The remaining candidates are then solved in $\mathcal{E}_{\text{syn}}$ using the same *search* and *browse* tools later exposed during reinforcement learning. A Judge agent evaluates the resulting trajectories in terms of answer correctness, total interaction steps, difficulty level, and shortcut risk, and returns a recommended action together with structured feedback to the Strategist for revision.

Let $g(x)$ denote the candidate distribution in-

duced by the generator, and let $v(x) \in \{0, 1\}$ indicate whether repeated solver trajectories satisfy $\bar{\psi}(x) \in \Omega$. The retained synthesis distribution is

$$p_{\text{syn}}(x) = \frac{g(x) v(x)}{\int g(x') v(x') dx'}. \quad (2)$$

Equation (2) shows that SYNSEARCH constructs the training-task distribution by filtering candidates according to behavioral compatibility with the target environment, rather than textual plausibility or nominal difficulty alone. The full synthesis procedure is given in Appendix B, and implementation details of filtering and validation are provided in Appendix C.

3.4 Multi-turn Reinforcement Learning

Given the verified synthesis distribution p_{syn} , we train a policy π_θ in $\mathcal{E}_{\text{syn}}$ by maximizing

$$J_{\text{RL}}(\theta) = \mathbb{E}_{x \sim p_{\text{syn}}} [\mathbb{E}_{\tau \sim p(\tau|x; \pi_\theta, \mathcal{E}_{\text{syn}})} [R(\tau, x)]] . \quad (3)$$

The reward is a composite of an outcome-level accuracy term and process-aware auxiliary terms:

$$R(\tau, x) = \sum_{k \in \{\text{acc}, \text{entity}, \text{uniq}, \text{decomp}\}} \lambda_k R_k(\tau, x), \quad (4)$$

with default weights $(\lambda_{\text{acc}}, \lambda_{\text{entity}}, \lambda_{\text{uniq}}, \lambda_{\text{decomp}}) = (0.5, 0.3, 0.1, 0.1)$. These auxiliary rewards are intentionally weak: they do not prescribe a fixed search strategy, but instead encourage non-trivial exploration and reduce degenerate behaviors such as redundant querying or shortcut-driven search. Appendix A provides basic properties of this bounded reward, and Appendix E summarizes hyperparameters and implementation details.

Answer correctness reward. The primary reward evaluates whether the final answer extracted from trajectory τ is equivalent to the reference answer $y^*(x)$, as judged by an LLM-based checker:

$$R_{\text{acc}}(\tau, x) = \mathbb{I}[\hat{y}(\tau) \equiv y^*(x)] . \quad (5)$$

Entity coverage reward. To encourage traversal of informative intermediate concepts, we reward coverage of bridge entities identified during task synthesis. Let $\mathcal{B}(x)$ denote the bridge-entity set for task x , and let $\mathcal{E}_{\text{match}}(\tau)$ denote the subset explicitly mentioned or unambiguously referenced in τ . We define

$$R_{\text{entity}}(\tau, x) = \frac{|\mathcal{E}_{\text{match}}(\tau) \cap \mathcal{B}(x)|}{|\mathcal{B}(x)| + \epsilon} . \quad (6)$$

Query uniqueness reward. To discourage redundant search, we reward trajectories whose queries are all distinct after normalization. Let $\mathcal{Q}(\tau) = (q_1, \dots, q_m)$ be the query sequence in τ , and let $\tilde{\mathcal{Q}}(\tau)$ be the corresponding normalized set. We define

$$R_{\text{uniq}}(\tau) = \mathbb{I}[|\tilde{\mathcal{Q}}(\tau)| = |\mathcal{Q}(\tau)|] . \quad (7)$$

Original-question avoidance reward. To encourage query decomposition rather than direct restatement, we penalize trajectories that issue the original question verbatim as a query. Let $\tilde{q}_{\text{orig}}(x)$ denote the normalized original question. We define

$$R_{\text{decomp}}(\tau, x) = \mathbb{I}[\tilde{q}_{\text{orig}}(x) \notin \tilde{\mathcal{Q}}(\tau)] . \quad (8)$$

Overall, SYNSEARCH couples *distribution design* with *policy optimization*. Task synthesis constructs an environment-aligned training distribution p_{syn} through trajectory-based validation, and reinforcement learning optimizes π_θ over that distribution in the same search–browse environment.

4 Experiments

4.1 Experimental Setup

Models and baselines. Our primary experiments use Qwen2.5-7B and Qwen2.5-3B, two representative model scales in recent open-source search-agent work (Jin et al., 2025; Sun et al., 2025a; Fan et al., 2025). We compare SYNSEARCH against the corresponding instruction-tuned base models and three strong baselines covering the most relevant training paradigms for our setting: Search-R1 (Jin et al., 2025), which trains in real search environments, and ZeroSearch (Sun et al., 2025a) and SSRL (Fan et al., 2025), which avoid real search APIs during training. For Qwen2.5-7B, we additionally include ASearcher (Gao et al., 2025), which explicitly studies synthetic data construction for deep search. To assess generalization beyond Qwen2.5, we also train and evaluate Llama3.1-8B and Qwen3-8B. Unless otherwise stated, all SYNSEARCH runs use Qwen3-235B-A22B-Instruct as the backend of the shared synthetic search–browse environment.

Training data. Unless otherwise specified, the main experiments use a set of 22,059 behaviorally validated training tasks synthesized by SYNSEARCH. These tasks are generated, filtered, and validated in the same synthetic search–browse environment later used for reinforcement learning,

and are retained only when they pass both quality filtering and solver-in-the-loop verification.

Interaction format and evaluation protocol. SYNSEARCH is trained in a structured tool-call format, whereas several baselines are released in the Search-R1-style textual ReAct format (Jin et al., 2025). To control for this difference, we report both formats for Qwen2.5-7B and Qwen2.5-3B whenever applicable. At evaluation time, all methods are executed under the same minimal ReAct-style outer loop: at each step, the model emits either a search action, a browse action, or a final answer, and the environment returns the corresponding observation. Search-R1-style agents serialize actions textually, whereas tool-call agents emit structured calls, but both follow the same alternating action–observation protocol. This evaluation setup isolates the effect of training while minimizing confounds from interface-specific execution logic. Additional implementation details, including failure handling and interface-specific execution notes, are given in Appendix E.

Benchmarks. We evaluate on both multi-hop QA and deep-search benchmarks. The multi-hop QA benchmarks are 2WikiMultiHopQA (Ho et al., 2020), Bamboogle (Press et al., 2023), HotpotQA (Yang et al., 2018), and MuSiQue (Trivedi et al., 2021); the deep-search benchmarks are GAIA (Mialon et al., 2023), WebWalker (Wu et al., 2025b), xbench (Chen et al., 2025a), and BrowseComp-zh (Wei et al., 2025a). For most benchmarks, we follow the evaluation suite released with Tool-Star (Dong et al., 2025); for xbench we use the full set of 100 questions, and for BrowseComp-zh we sample 200 questions to match a comparable evaluation scale. All methods are evaluated under the same sample selection and judging rules.

Metrics. Our primary metric is answer accuracy, measured by semantic equivalence under an LLM-based evaluator (Kimi-K2.5; human spot-check in Appendix G), depending on the benchmark. To assess learned search behavior, we also report average search steps (AVGSEARCH), average browse steps (AVGBROWSE), the per-question duplicate-query ratio (PERQDUP), and the rate of issuing the original question directly as a search query (ORIGQ). When useful, we additionally report averages over the multi-hop QA benchmarks, the deep-search benchmarks, and the full benchmark suite. We use answer accuracy as the primary criterion for model comparison and treat

the behavioral metrics as complementary indicators of the search strategies acquired during training. Unless otherwise specified, reported numbers are means over 4 independent evaluation runs of the same checkpoint. Formally, for a policy π and evaluation set $\mathcal{D}_{\text{eval}}$, accuracy is defined as $\text{Acc}(\pi) = \frac{1}{|\mathcal{D}_{\text{eval}}|} \sum_{x \in \mathcal{D}_{\text{eval}}} \mathbb{1}[\hat{y}_{\pi}(x) \equiv y^*(x)]$.

4.2 Main Results

Tables 1 and 2 report results under Search-R1-style interaction and structured tool-call interaction, respectively. Under the textual protocol, SYNSEARCH achieves the best average at both model scales. The gains are especially clear on deep-search benchmarks: the deep-search average rises to 26.23 on Qwen2.5-7B and 21.15 on Qwen2.5-3B, compared with 22.90 and 18.28 for SSRL, respectively. Under structured tool-call interaction, SYNSEARCH outperforms all compared baselines at both model scales, indicating that the gains do not stem from a particular interaction format. SYNSEARCH remains effective under both interaction formats. Overall, SYNSEARCH improves answer quality across model scales and interaction formats. This pattern supports our hypothesis that environment-aligned, trajectory-validated tasks induce stronger deep-search policies.

4.3 Backbone Generalization

To test whether the gains of SYNSEARCH extend beyond Qwen2.5, we evaluate Llama3.1-8B and Qwen3-8B under structured tool-call interaction. Table 3 shows consistent improvements on both backbones. For Llama3.1-8B, the overall average improves from 31.29 to 41.16, while the deep-search average rises from 13.13 to 28.60. For Qwen3-8B, the overall average improves from 40.31 to 50.31, with the deep-search average increasing from 24.50 to 35.42. The gains are again largest on the deep-search benchmarks, suggesting that the benefits of SYNSEARCH are not tied to a specific backbone family.

4.4 Ablation Studies

We ablate SYNSEARCH on Qwen2.5-7B to isolate the effects of the training-task distribution, the synthesis pipeline, and the reward design.

Training-task distribution. We compare three training sets of matched size under the same backbone and protocol: HotpotQA, ASearcher, and our environment-aligned tasks. Table 5 shows that our data yields the best overall average (44.61),

Method	Multi-hop QA					Deep Search					Avg.
	2Wiki	Bamboogle	HQA	MuSiQ	MH Avg.	GAIA	WebWalker	xbench	BrowseComp-zh	DS Avg.	
Qwen2.5-7B-Instruct											
Qwen2.5-7B-Instruct	50.5	57.6	51.0	22.0	45.28	<u>12.6</u>	24.0	34.0	4.0	18.65	31.96
Search-R1	57.0	65.6	52.5	21.5	49.15	10.7	13.5	17.0	2.5	10.93	30.04
ZeroSearch	51.0	46.4	52.5	21.5	42.85	11.2	23.0	47.0	3.5	21.18	32.01
SSRL	56.5	58.4	53.5	25.0	48.35	12.1	<u>25.5</u>	<u>48.5</u>	<u>5.5</u>	<u>22.90</u>	<u>35.63</u>
ASearcher	60.0	<u>68.0</u>	<u>54.5</u>	<u>26.5</u>	<u>52.25</u>	11.7	15.5	17.0	4.5	12.18	32.21
Ours	<u>59.0</u>	68.8	57.0	27.0	52.95	18.4	31.5	49.0	6.0	26.23	39.59
Qwen2.5-3B-Instruct											
Qwen2.5-3B-Instruct	33.5	40.0	38.5	8.5	30.12	6.8	17.0	33.0	<u>3.0</u>	14.95	22.54
Search-R1	38.0	24.8	52.0	11.0	31.45	9.7	17.5	<u>41.0</u>	2.5	17.68	24.56
ZeroSearch	37.0	39.2	46.5	17.5	35.05	13.6	18.5	37.0	1.0	17.53	26.29
SSRL	44.5	<u>40.8</u>	48.5	<u>19.0</u>	<u>38.20</u>	<u>14.1</u>	<u>22.0</u>	35.0	2.0	<u>18.28</u>	<u>28.24</u>
Ours	<u>43.0</u>	43.2	<u>51.0</u>	21.0	39.55	14.6	24.0	42.0	4.0	21.15	30.35

Table 1: **Main results under Search-R1-style textual interaction.** LLM-judge answer accuracy (%) on Qwen2.5-7B and Qwen2.5-3B. All numbers are means over 4 independent evaluation runs of the same checkpoint. Bold and underlined values denote the best and second-best scores per column.

Method	Multi-hop QA					Deep Search					Avg.
	2Wiki	Bamboogle	HQA	MuSiQ	MH Avg.	GAIA	WebWalker	xbench	BrowseComp-zh	DS Avg.	
Qwen2.5-7B-Instruct											
Qwen2.5-7B-Instruct	52.0	<u>50.4</u>	<u>56.5</u>	22.0	<u>45.23</u>	<u>18.4</u>	24.0	51.0	4.0	<u>24.35</u>	34.79
Search-R1	46.0	52.0	46.5	18.0	40.63	9.7	18.0	33.0	3.5	16.05	28.34
ZeroSearch	<u>53.5</u>	47.2	53.0	<u>22.5</u>	44.05	12.6	<u>28.0</u>	45.0	5.0	22.65	33.35
SSRL	52.5	50.8	54.0	20.0	44.33	16.5	23.0	<u>52.0</u>	<u>5.5</u>	24.25	34.29
ASearcher	51.5	55.2	49.0	21.0	44.18	11.7	21.5	39.0	4.0	19.05	31.62
Ours	62.5	65.6	66.0	29.5	55.90	24.3	34.0	69.0	6.0	33.33	44.61
Qwen2.5-3B-Instruct											
Qwen2.5-3B-Instruct	<u>44.0</u>	<u>42.4</u>	43.0	15.0	<u>36.10</u>	<u>8.7</u>	17.0	15.0	1.0	10.43	<u>23.26</u>
Search-R1	27.0	20.8	17.0	10.0	18.70	6.8	6.0	12.0	3.0	6.95	12.83
ZeroSearch	31.5	24.8	<u>44.0</u>	15.5	28.95	7.8	<u>22.5</u>	<u>39.0</u>	1.0	<u>17.58</u>	<u>23.26</u>
SSRL	36.5	33.6	38.5	16.5	31.28	4.9	14.0	14.0	2.5	8.85	20.06
Ours	54.5	56.8	59.5	23.5	48.58	21.4	28.0	42.0	3.5	23.73	36.15

Table 2: **Main results under structured tool-call interaction.** LLM-judge answer accuracy (%) on Qwen2.5-7B and Qwen2.5-3B. All numbers are means over 4 independent evaluation runs of the same checkpoint. Bold and underlined values denote the best and second-best scores in each column.

compared with 41.16 for HotpotQA and 35.49 for ASearcher. It also induces substantially more browsing than ASearcher (1.70 vs. 0.01), suggesting that generic multi-hop or synthetic QA data alone is insufficient to train browse-intensive deep-search behavior.

Task-synthesis pipeline. We next ablate the synthesis pipeline. CORE uses multi-agent rewriting only, CORE+QG adds Quality Gate filtering, and **Full** further adds trajectory-based validation and final verification. As shown in Table 6, performance improves steadily from 39.33 to 41.29 to 44.61. Notably, CORE produces many search steps but almost no browsing (4.43 / 0.07), indicating that rewriting alone can create superficially difficult but browse-poor tasks. Both filtering and trajectory-based validation are therefore important for constructing behaviorally useful training data.

Process-aware rewards. Finally, we compare answer-only training with the full reward. Table 7 shows that process-aware rewards improve the overall average, while also shifting behavior in the intended direction: browsing increases (0.41→1.70), duplicate querying decreases (8.98%→7.18%), and direct reuse of the original question as a query drops sharply (43.99%→1.02%). Thus, the auxiliary rewards improve not only final accuracy but also the learned search process.

4.5 Discussion

Environment-aligned task distributions matter. The gains of SYNSEARCH come not from synthetic data alone, but from constructing a training distribution aligned with the target search-browse environment. Under matched backbones and training settings, our synthesized tasks outperform both

Backbone	Method	Multi-hop QA					Deep Search					Avg.
		2Wiki	Bamboogle	HQA	MuSiQ	MH Avg.	GAIA	WebWalker	xbench	BrowseComp-zh	DS Avg.	
Llama3.1-8B	Base	55.5	60.8	58.5	23.0	49.45	17.5	20.5	13.0	1.5	13.13	31.29
	Ours	59.0	66.4	64.5	25.0	53.73	19.4	35.0	54.0	6.0	28.60	41.16
Qwen3-8B	Base	66.0	68.0	61.0	29.5	56.12	18.5	34.5	40.0	5.0	24.50	40.31
	Ours	81.0	76.8	71.0	32.0	65.20	28.2	42.5	64.0	7.0	35.42	50.31

Table 3: **Backbone generalization under structured tool-call interaction.** LLM-judge answer accuracy (%) on two additional 8B backbones. All numbers are means over 4 independent evaluation runs of the same checkpoint.

Model	Multi-hop QA					Deep Search					Avg.
	2Wiki	Bamboogle	HQA	MuSiQ	MH Avg.	GAIA	WebWalker	xbench	BrowseComp-zh	DS Avg.	
Qwen2.5-7B + synthetic env	48.5	44.0	38.0	19.5	37.50	5.8	5.0	6.0	3.5	5.08	21.29
Ours-7B + synthetic env	47.0	69.6	46.5	30.5	48.40	10.7	6.5	14.0	7.5	9.68	29.04
Qwen2.5-7B + real-search env	52.0	50.4	56.5	22.0	45.23	18.4	24.0	51.0	4.0	24.35	34.79
Ours-7B + real-search env	62.5	65.6	66.0	29.5	55.90	24.3	34.0	69.0	6.0	33.33	44.61

Table 4: **Evaluation under synthetic and real-search environments.** LLM-judge answer accuracy (%) for the base model and our trained model when executed in the synthetic and real-search environments.

Data	MH	DS	Avg.	Search	Browse
Base	45.23	24.35	34.79	1.84	1.44
HotpotQA	52.73	29.60	41.16	2.63	0.87
ASearcher	45.35	25.63	35.49	1.35	0.01
Ours	55.90	33.33	44.61	4.00	1.70

Table 5: **Effect of training-task distribution on Qwen2.5-7B.** Training sets of matched size under the same backbone and training protocol.

Variant	MH	DS	Avg.	Search	Browse
Base	45.23	24.35	34.79	1.84	1.44
CORE	51.33	27.33	39.33	4.43	0.07
CORE+QG	52.73	29.85	41.29	2.62	0.87
Full	55.90	33.33	44.61	4.00	1.70

Table 6: **Task-synthesis ablation on Qwen2.5-7B.** QG denotes Quality Gate.

benchmark-derived and prior synthetic data, while inducing richer behavior, especially more browsing. For deep-search agents, the task distribution should therefore be treated as a first-class design object rather than inherited from existing QA resources.

Deep search requires shaping behavior, not just answers. Our reward ablation shows that final accuracy is only part of the story. Process-aware rewards increase browsing, reduce duplicate querying, and discourage directly restating the original question as a search query. These changes matter because deep search is fundamentally sequential: shortcut answers do not substitute for learning to decompose questions, traverse intermediate entities, and inspect retrieved pages.

What the synthetic environment captures. Our

Metric	Answer-only	Ours
MH Avg. (%)	50.98	55.90
DS Avg. (%)	29.85	33.33
Avg.	40.41	44.61
AvgSearch	2.13	4.00
AvgBrowse	0.41	1.70
PerQDup (%)↓	8.98	7.18
OrigQ (%)↓	43.99	1.02

Table 7: **Reward ablation on Qwen2.5-7B.** Answer-only training versus the full reward.

synthetic environment is designed for task construction and training rather than as a substitute for real-search evaluation, and Table 4 reports two observations that support this design. From the base to our trained model, accuracy improves substantially under both synthetic and real-search evaluation, indicating that the policy is not overfitting to a particular retrieval backend but acquiring transferable deep-search competence. Moreover, on several fact-stable benchmarks such as Bamboogle and MuSiQue, the trained model achieves comparable performance in the synthetic and real-search environments, consistent with the synthetic backend already encoding the stable, well-established facts these benchmarks rely on, and supporting the feasibility of LLM-backed simulators for training-time interaction.

5 Conclusion

We presented SYNSEARCH, a framework for training deep-search agents without calling real web APIs during training, by jointly designing the training-task distribution and the policy-learning

environment in a single closed loop. The synthetic search–browse environment serves as a controllable, reproducible, and low-marginal-cost interface for training, while real search remains the yardstick for evaluation. Across model families, benchmarks, and interaction formats, SYNSEARCH improves both answer quality and search behavior, with especially strong gains on deep-search benchmarks. Our results suggest that progress in deep-search training depends not only on better optimization, but also on a shared-environment closed loop that couples task synthesis with policy learning. The resulting agents achieve strong performance under real-search evaluation.

Limitations

Although SYNSEARCH improves performance under both synthetic and real-search evaluation, a substantial gap remains between the two. The synthetic environment approximates the interaction interface of search and browsing tools, but does not fully capture the recall properties, temporal drift, document heterogeneity, or evidence structure of the open web. It should therefore be viewed as a practical and reproducible training environment rather than a full surrogate for real search.

Our synthesis pipeline also depends on solver-in-the-loop validation, judge feedback, and heuristic filtering. These mechanisms improve task quality, but they also shape the final training distribution, since validity is partly defined relative to the capabilities and biases of the solver and judge used during synthesis. Developing more diverse and less model-dependent validation strategies is an important direction for future work.

Our reported numbers are means over four evaluation runs of a single trained checkpoint, which captures decoding and environment variability but not variance across independently trained policies. Since the method combines GRPO sampling with generated tasks, results from separate training runs may differ, and we do not report seed-level confidence intervals. The consistency of the gains across four backbones provides indirect evidence that the improvements are not specific to one run, but it does not replace a multi-seed study under a fixed backbone.

The synthetic environment is driven by a large instruction-tuned backend (Qwen3-235B-A22B-Instruct). Although its per-call cost is low and cached responses are reused across runs, serving a

model of this size still requires substantial accelerator capacity, which limits accessibility for smaller compute budgets. We did not vary the backend scale, so it remains open how far synthesis and training quality degrade when a smaller model acts as the search and browse backend.

Ethical Considerations

The synthetic environment used in SYNSEARCH inherits the knowledge distribution, factual errors, omissions, and social biases of its underlying LLM backend. As a result, synthesized tasks and trained policies may reflect these imperfections. Additional auditing, factuality checks, and bias evaluation are therefore desirable, especially before real-world deployment.

Coverage is also uneven. Because the backend is a single model, the facts it can simulate are denser for widely documented, English-centric and high-resource topics than for other languages, regions, and long-tail entities. Both the synthesized tasks and the resulting policies are therefore likely to work less well outside that core, and aggregate accuracy can mask this imbalance. Reporting results by language and region, and seeding synthesis from more diverse sources, would make the gap visible rather than implicit.

Our real-search evaluation, unlike training, retrieves and parses live web pages. Fetched content may include personal information or copyrighted material, and we use it only to score answers during evaluation: pages are parsed to text, truncated, used in-context, and not redistributed or used to train models. Anyone reproducing this setup should follow the terms of service of the search provider and of the sites being fetched, and respect the usual crawling conventions.

More broadly, our goal is to enable low-cost and reproducible study of deep-search training rather than to replace deployment-time evaluation. In practical or higher-stakes settings, simulator-trained agents should be complemented by benchmark testing, monitoring, and appropriate human oversight.

Acknowledgments

This work was supported by Alibaba Innovative Research Program.

References

- Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. 2024. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, volume 2024, pages 9112–9141.
- Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. 2021. Improving language models by retrieving from trillions of tokens. *arXiv preprint arXiv:2112.04426*.
- Kaiyuan Chen, Yixin Ren, Yang Liu, Xiaobo Hu, Hao-tong Tian, Tianbao Xie, Fangfu Liu, Haoye Zhang, Hongzhang Liu, Yuan Gong, Chen Sun, Han Hou, Hui Yang, James Pan, Jianan Lou, Jiayi Mao, Jizheng Liu, Jinpeng Li, Kangyi Liu, Kenkun Liu, Rui Wang, Run Li, Tong Niu, Wenlong Zhang, Wenqi Yan, Xuanzheng Wang, Yuchen Zhang, Yi-Hsin Hung, Yuan Jiang, Zexuan Liu, Zihan Yin, Zijian Ma, and Zhiwen Mo. 2025a. *xbench: Tracking agents productivity scaling with profession-aligned real-world evaluations*. *arXiv preprint arXiv:2506.13651*.
- Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z Pan, Wen Zhang, Huajun Chen, et al. 2025b. Learning to reason with search for llms via reinforcement learning. *arXiv preprint arXiv:2503.19470*.
- Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. 2025. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*.
- Yuchen Fan, Kaiyan Zhang, Heng Zhou, Yuxin Zuo, Yanxu Chen, Yu Fu, Xinwei Long, Xuekai Zhu, Che Jiang, Yuchen Zhang, et al. 2025. Ssr1: Self-search reinforcement learning. *arXiv preprint arXiv:2508.10874*.
- Jiaxuan Gao, Wei Fu, Mingyang Xie, Shusheng Xu, Chuyi He, Zhiyu Mei, Banghua Zhu, and Yi Wu. 2025. Beyond ten turns: Unlocking long-horizon agentic search with large-scale asynchronous rl. *arXiv preprint arXiv:2508.07976*.
- Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2022. Precise zero-shot dense retrieval without relevance labels. *arXiv preprint arXiv:2212.10496*.
- Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*.
- 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. International Committee on Computational Linguistics.
- Yuxuan Huang, Yihang Chen, Haozheng Zhang, Kang Li, Huichi Zhou, Meng Fang, Linyi Yang, Xiaoguang Li, Lifeng Shang, Songcen Xu, et al. 2025. Deep research agents: A systematic examination and roadmap. *arXiv preprint arXiv:2506.18096*.
- Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Atlas: Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*.
- Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. *arXiv preprint arXiv:2305.06983*.
- Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Serkan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-rl: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.
- Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025a. 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. 2025b. Search-ol: Agentic search-enhanced large reasoning models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5420–5438.
- Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. 2025c. Webthinker: Empowering large reasoning models with deep research capability. *arXiv preprint arXiv:2504.21776*.
- Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.
- Yuanjie Lyu, Chengyu Wang, Lei Shen, Jun Huang, and Tong Xu. 2026a. Mock worlds, real skills: Building small agentic language models with synthetic tasks, simulated environments, and rubric-based rewards. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12529–12545.

- Yuanjie Lyu, Chengyu Wang, Haonan Zheng, Yuanhao Yue, Junbing Yan, Ming Wang, and Jun Huang. 2026b. Agenticqwen: Training small agentic language models with dual data flywheels for industrial-scale tool use. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL 2026)*, pages 535–551.
- Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. *Gaia: A benchmark for general ai assistants*. *arXiv preprint arXiv:2311.12983*.
- Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.
- Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. 2023. *Measuring and narrowing the compositionality gap in language models*. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711. Association for Computational Linguistics.
- Yujia Qin, Binyuan Liang, Yanheng Ye, Ningyu Xu, Sheng Chen, Zhiyuan Tan, Xiubo Huang, Quoc Li, Tao Huang, Ningyu Zheng, et al. 2023. *Toolllm: Facilitating large language models to master 16000+ real-world apis*. *arXiv preprint arXiv:2307.16789*.
- Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, et al. 2025. Taskcraft: Automated generation of agentic tasks. *arXiv preprint arXiv:2506.10055*.
- Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2023. *Replug: Retrieval-augmented black-box language models*. *arXiv preprint arXiv:2301.12652*.
- Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Jirong 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. 2025a. Zeroset: Incentivize the search capability of llms without searching. *arXiv preprint arXiv:2505.04588*.
- Shuang Sun, Huatong Song, Yuhao Wang, Ruiyang Ren, Jinhao Jiang, Junjie Zhang, Fei Bai, Jia Deng, Wayne Xin Zhao, Zheng Liu, et al. 2025b. Simpledeepsearcher: Deep information seeking via web-powered reasoning trajectory synthesis. *arXiv preprint arXiv:2505.16834*.
- Zhiwen Tan, Jiaming Huang, Qintong Wu, Hongxuan Zhang, Chenyi Zhuang, and Jinjie Gu. 2026. Rag-r1: Incentivizing the search and reasoning capabilities of llms through multi-query parallelism. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 33187–33195.
- Zhengwei Tao, Jialong Wu, Wenbiao Yin, Junkai Zhang, Baixuan Li, Haiyang Shen, Kuan Li, Liwen Zhang, Xinyu Wang, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025. *Webshaper: Agentic data synthesizing via information-seeking formalization*. *arXiv preprint arXiv:2507.15061*.
- Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2021. *MuSiQue: Multi-hop questions via single-hop question composition*. *CoRR*, abs/2108.00573.
- Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. *Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions*. *arXiv preprint arXiv:2212.10509*.
- Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khoshdel, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 13484–13508.
- Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025a. *Browsecomp: A simple yet challenging benchmark for browsing agents*. *arXiv preprint arXiv:2504.12516*.
- Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. 2025b. *WebAgent-r1: Training web agents via end-to-end multi-turn reinforcement learning*. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7909–7928, Suzhou, China. Association for Computational Linguistics.
- Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Gang Fu, Yong Jiang, et al. 2025a. *Webdancer: Towards autonomous information seeking agency*. *arXiv preprint arXiv:2505.22648*.
- Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and Fei Huang. 2025b. *WebWalker: Benchmarking LLMs in web traversal*. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10290–10305, Vienna, Austria. Association for Computational Linguistics.
- Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2025. *Wizardlm: Empowering large pre-trained language models to follow complex instructions*.

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](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. 2022. [Generate rather than retrieve: Large language models are strong context generators](#). *arXiv preprint arXiv:2209.10063*.

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. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 414–431.

A Additional Theoretical Details

This appendix records basic properties of the validity-filtered synthesis distribution and the bounded composite reward used in SYNSEARCH.

A.1 Properties of the Validity-Filtered Synthesis Distribution

Recall that the candidate generator induces a distribution g over the task space $\mathcal{X}$, and that the validity indicator $v(x) \in \{0, 1\}$ specifies whether task x satisfies the trajectory-based validity criterion. The retained synthesis distribution is

$$p_{\text{syn}}(x) = \frac{g(x) v(x)}{\int g(x') v(x') dx'}. \quad (9)$$

Proposition 1 (Support of the retained synthesis distribution). *Assume the normalizing constant $Z = \int g(x') v(x') dx'$ is strictly positive. Then the support of p_{syn} is contained in the valid-task set $\mathcal{X}_{\text{valid}}$, i.e., $p_{\text{syn}}(x) > 0 \implies x \in \mathcal{X}_{\text{valid}}$.*

Proof. If $p_{\text{syn}}(x) > 0$, then Eq. (9) implies $g(x) v(x) > 0$. Since $v(x) \in \{0, 1\}$, we must have $v(x) = 1$. By definition, this means that the repeated solver trajectories for x satisfy the validity condition, so $x \in \mathcal{X}_{\text{valid}}$. $\square$

Proposition 2 (Behavioral profile as a conditional expectation). *Let $\phi(x, \mathcal{E})$ denote the behavioral descriptor. Under the same assumptions as Proposition 1, $\Phi(p_{\text{syn}}, \mathcal{E}) = \mathbb{E}_{x \sim g}[\phi(x, \mathcal{E}) \mid v(x) = 1]$.*

Proof. By definition, $\Phi(p_{\text{syn}}, \mathcal{E}) = \int \phi(x, \mathcal{E}) p_{\text{syn}}(x) dx$. Substituting Eq. (9) gives $\Phi(p_{\text{syn}}, \mathcal{E}) = \int \phi(x, \mathcal{E}) \frac{g(x) v(x)}{Z} dx$, where $Z = \int g(x') v(x') dx' = \Pr_{x \sim g}[v(x) = 1]$. Since $v(x)$ is the indicator of the validity event, the result is exactly the conditional expectation of $\phi(x, \mathcal{E})$ under g given $v(x) = 1$. $\square$

Proposition 3 (Monotone validity filtering). *Let $v_1, v_2 : \mathcal{X} \rightarrow \{0, 1\}$ satisfy $v_2(x) \leq v_1(x)$ for all $x \in \mathcal{X}$. Then the support of the retained distribution under v_2 is contained in that under v_1 .*

Proof. Let $p_i(x) = \frac{g(x) v_i(x)}{\int g(x') v_i(x') dx'}$ for $i \in \{1, 2\}$, assuming both normalizing constants are positive. If $p_2(x) > 0$, then $g(x) v_2(x) > 0$, so $v_2(x) = 1$. Since $v_2(x) \leq v_1(x)$ and both indicators are binary, we also have $v_1(x) = 1$. Hence $g(x) v_1(x) > 0$, which implies $p_1(x) > 0$. Therefore $\text{supp}(p_2) \subseteq \text{supp}(p_1)$. $\square$

A.2 Properties of the Composite Reward

Recall the composite reward

$$R(\tau, x) = \sum_{k \in \{\text{acc}, \text{entity}, \text{uniq}, \text{decomp}\}} \lambda_k R_k(\tau, x), \quad (10)$$

where $\lambda_k \geq 0$ and, by default, $\sum_k \lambda_k = 1$.

Proposition 4 (Boundedness of the composite reward). *Assume $\lambda_k \geq 0$ for all components and $\sum_k \lambda_k = 1$. If each component reward satisfies $R_k(\tau, x) \in [0, 1]$, then the composite reward in Eq. (10) also satisfies $R(\tau, x) \in [0, 1]$.*

Proof. Since each $R_k(\tau, x) \in [0, 1]$ and each $\lambda_k \geq 0$, we have $0 \leq \lambda_k R_k(\tau, x) \leq \lambda_k$ for every component k . Summing over all components gives $0 \leq \sum_k \lambda_k R_k(\tau, x) \leq \sum_k \lambda_k = 1$, so $R(\tau, x) \in [0, 1]$. $\square$

Proposition 5 (Reward monotonicity with respect to entity coverage). *Fix a task x and assume the bridge-entity set $\mathcal{B}(x)$ is nonempty. For two trajectories τ_1 and τ_2 , if $\mathcal{E}_{\text{match}}(\tau_1) \cap \mathcal{B}(x) \subseteq \mathcal{E}_{\text{match}}(\tau_2) \cap \mathcal{B}(x)$, then $R_{\text{entity}}(\tau_1, x) \leq R_{\text{entity}}(\tau_2, x)$.*

Proof. By definition, $R_{\text{entity}}(\tau, x) = \frac{|\mathcal{E}_{\text{match}}(\tau) \cap \mathcal{B}(x)|}{|\mathcal{B}(x)| + \epsilon}$. The denominator is fixed for task x and strictly positive. The assumed set inclusion implies $|\mathcal{E}_{\text{match}}(\tau_1) \cap \mathcal{B}(x)| \leq |\mathcal{E}_{\text{match}}(\tau_2) \cap \mathcal{B}(x)|$, and the claim follows. $\square$

A.3 Remarks

These propositions record three basic properties used in SYNSEARCH: p_{syn} assigns mass only to trajectory-valid tasks, validity filtering can be viewed as conditioning on a behavioral event, and the composite reward remains bounded under non-negative convex weighting.

B Algorithmic Details

Algorithm 1 summarizes the closed-loop task synthesis procedure used by SYNSEARCH. The procedure maintains both a current state and the last verified valid state, which enables rollback when a proposed modification fails quality filtering or trajectory-based validation in the same synthetic environment later used for policy learning.

C Instantiation of Task Validity and Judge Criteria

The validity condition is defined abstractly through the admissible region $\Omega \subseteq \mathbb{R}^d$. In practice, Ω is instantiated through thresholded criteria computed from repeated solver trajectories and judge outputs.

Trajectory-level signals. For each candidate, the Judge agent analyzes the solver’s full interaction trace and returns a structured assessment comprising the following fields:

- **Answer correctness** (`is_correct`): whether the solver’s final answer is semantically equivalent to the reference;
- **Total steps** (`total_steps`): the number of search and browse actions executed before the solver terminates;
- **Difficulty level** (`difficulty_level`): a categorical label, with choices *too_easy* (fewer than 4 steps or an obvious shortcut), *good* (4+ steps with a coherent multi-hop path), *too_hard* (many steps but incorrect), or *broken* (logical or factual issues);
- **Shortcut risk** (`has_shortcut`): whether the task can be solved through direct lexical overlap, snippet-only matching, or superficial reformulation of the original question;
- **Recommended action** (`recommended_action`): one of EXPAND, REFINE, ROLLBACK, or FINALIZE, which is fed back to the Strategist to guide the next iteration.

Algorithm 1 Environment-Aligned Task Synthesis

Require: Seed task $x^{(0)}$, synthetic environment $\mathcal{E}_{\text{syn}}$, maximum iterations T , number of solver rollouts K

Ensure: Verified task set $\mathcal{D}$

```

1:  $\mathcal{D} \leftarrow \emptyset$ 
2: Initialize current state  $s \leftarrow x^{(0)}$ 
3: Initialize last valid state  $s_{\text{valid}} \leftarrow \emptyset$ 
4: for  $k = 1$  to  $T$  do
5:   Select meta-action  $m_k \in \{\text{EXPAND, REFINE, ROLLBACK, FINALIZE}\}$ 
6:   if  $m_k = \text{ROLLBACK}$  then
7:     if  $s_{\text{valid}} \neq \emptyset$  then
8:        $s \leftarrow s_{\text{valid}}$ 
9:     end if
10:    continue
11:  end if
12:  if  $m_k = \text{FINALIZE}$  then
13:    if  $s_{\text{valid}} \neq \emptyset$  then
14:      Export Task( $s_{\text{valid}}$ ) to  $\mathcal{D}$ 
15:    end if
16:    break
17:  end if
18:  Generate candidate state  $\tilde{s}$  from  $s$  using meta-action  $m_k$ 
19:  if QUALITYGATE( $\tilde{s}$ ) = FAIL then
20:    continue
21:  end if
22:  Collect solver trajectories  $\mathcal{T}(\tilde{s}) = \{\tau^{(1)}, \dots, \tau^{(K)}\}$ 
23:  in  $\mathcal{E}_{\text{syn}}$ 
24:  Compute judge report  $\rho \leftarrow \text{JUDGE}(\mathcal{T}(\tilde{s}))$ 
25:  if VALID( $\rho$ ) then
26:     $s \leftarrow \text{UPDATE}(\tilde{s}, \rho)$ 
27:     $s_{\text{valid}} \leftarrow s$ 
28:  else
29:    if  $s_{\text{valid}} \neq \emptyset$  then
30:       $s \leftarrow s_{\text{valid}}$ 
31:    end if
32:  end if
33: end for
34: for all  $x \in \mathcal{D}$  do
35:   Collect repeated solver trajectories  $\mathcal{T}(x)$  in  $\mathcal{E}_{\text{syn}}$ 
36:   if STABLEVALID( $\mathcal{T}(x)$ ) = FALSE then
37:     Remove  $x$  from  $\mathcal{D}$ 
38:   end if
39: end for
40: return  $\mathcal{D}$ 

```

Single-run validity gate. A candidate task passes the validity gate if and only if `is_correct` = TRUE and `difficulty_level` = *good*. Tasks that fail this check are either rolled back or discarded.

Repeated-rollout stability check. Candidates that pass the single-run validity gate are further evaluated through $K=4$ independent solver rollouts in $\mathcal{E}_{\text{syn}}$. Each rollout records the predicted answer and the number of interaction turns. We compute the answer accuracy (fraction of runs with a correct answer) and average turn count across the K runs. This repeated evaluation serves as a stability annotation: it quantifies how reliably the task elicits correct multi-step solutions rather than

depending on a single lucky trajectory.

Why repeated validation? Repeated-rollout validation reduces false positives from single lucky trajectories and provides a signal of task stability. Tasks with high accuracy and consistent turn counts across runs are more likely to provide a reliable training signal during reinforcement learning.

Behavioral interpretation. A task is useful for training not merely because it is linguistically plausible, but because it reliably elicits the search–browse behavior that the target policy is intended to learn.

Judge rubrics. The Judge agent receives the solver’s full interaction trajectory (truncated to 15K characters) and returns a compact JSON assessment. The exact prompt and output schema are given in Appendix F (Figure 5).

D Implementation Details for the Shared Synthetic Environment

We summarize key implementation details of the shared synthetic environment $\mathcal{E}_{\text{syn}}$. Unless otherwise stated, the same environment API is used during task synthesis and reinforcement learning.

Environment backend. The shared environment is instantiated with Qwen3-235B-A22B-Instruct. We use a fixed backend within each experiment to keep task synthesis, trajectory validation, and policy learning aligned. Additional backend configuration details, including decoding settings and system prompts, are given in Appendix F.

Search interface. The search tool takes a query string as input and returns a ranked list of candidates, each consisting of (title, url, snippet) triples, with up to $N=5$ results per query. The exact prompting template, ranking format, and truncation rules are given in Appendix F.

Browse interface. The browse tool takes a target page or document identifier together with optional context of the originating search query and the selected search result snippet and returns a tuple $(\text{url}, \text{title}, \text{page_content})$, where page content is truncated to at most 4096 tokens. We use context-linked browsing so that returned content remains anchored to the originating query and search result.

Environment reuse across stages. The same environment interface is reused during task synthesis and reinforcement learning, reducing mismatch between validation and training trajectories. This shared setup is central to SYNSEARCH, since tasks are generated, filtered, and later optimized in the same synthetic search–browse environment.

Leakage control. Search and browse prompts are restricted to information naturally available at inference time. In particular, they do not condition on hidden references, gold answers, or any task meta-data unavailable to the policy at execution time. The exact prompt templates and leakage-control checks are given in Appendix F.

Context and history handling. The environment maintains a growing list of (role, content) message pairs following the standard chat format, where tool responses are appended as user-role messages wrapped in `<tool_response>` tags across search and browse steps. We use a maximum prompt length of 8192 tokens; tool responses are individually truncated to at most 4096 tokens before appending to the history to control context length and ensure stable execution.

Failure handling. When the backend produces malformed outputs, empty returns, or other execution failures, the environment returns a structured JSON error message (e.g., `{"error": "..."}`) as the tool response and the episode continues; if the model output is entirely unparseable, the episode terminates immediately. The exact handling rules used in evaluation are described in Appendix E.

E Reproducibility Details

This appendix summarizes the main implementation settings, hyperparameters, and evaluation details used in our experiments.

E.1 Synthesis Configuration

Table 8 summarizes the main configuration used in environment-aligned task synthesis.

E.2 Training Hyperparameters

Table 9 summarizes the main reinforcement-learning settings.

E.3 Synthetic Environment Settings

Table 10 summarizes the main settings of the shared synthetic search–browse environment.

Item	Value
Seed-task source	HotpotQA medium bridge
Maximum synthesis iterations T	20
Solver rollouts per candidate K	4
Strategist model	Kimi-K2.5
Solver model used in validation	Qwen3-235B-A22B-Instruct
Judge model	Kimi-K2.5
Quality Gate criteria	Uniqueness, Pseudo-multi-hop, Answer leakage
Validity thresholds	is_correct=True AND difficulty_level="good"
Stable-valid thresholds	4-run final solver evaluation with accuracy and avg_turns

Table 8: **Task-synthesis configuration.** Main settings for closed-loop task synthesis and trajectory-based validation.

Item	Value
Training set size	22,059 validated tasks
RL algorithm	GRPO
Policy backbone(s)	Qwen2.5-7B-Instruct (for most experiments)
Learning rate	1e-6
Batch size	64
Mini-batch size	16
Number of training steps	3 epochs
Rollouts per update	8
Maximum turns per episode	15
Reward weights (λ_{acc} , λ_{entity} , λ_{uniq} , λ_{decomp})	(0.5, 0.3, 0.1, 0.1)
Optimizer	Adam
Computing infrastructure	8× NVIDIA H20-96GB
Approximate training time	~20 hours (per model)

Table 9: **Training hyperparameters.** Main settings for multi-turn reinforcement learning.

E.4 Evaluation Protocol Details

Unless otherwise stated, all methods are evaluated under the same minimal ReAct-style outer loop. At each step, the model emits either a `search` action, a `browse` action, or a final answer, and the environment returns the corresponding observation.

Step limits and stopping. We use at most 15 tool-interaction steps per episode. An episode terminates when the model emits a final answer or reaches the maximum step limit.

Action parsing. For structured tool-call agents, model outputs are parsed into executable calls according to a fixed schema. For Search-R1-style textual agents, actions are extracted from text using rule-based parsing under the same outer-loop controller.

Invalid-action and execution-failure handling. If a model output cannot be parsed into a valid action (i.e., it contains neither a recognizable tool call nor a final answer), the episode is terminated immediately and the step is counted as a wasted terminal action with a fallback answer. No retry is attempted for unparseable outputs. If an action is syntactically valid but semantically unusable (e.g., empty query, malformed URL, unknown tool name, or a runtime

execution error), the environment returns a structured error message as the tool response and the episode continues, allowing the model to recover in subsequent steps.

Stable executability under structured tool-call interaction. All baselines are evaluated end-to-end under the same outer-loop controller. For methods originally released in textual ReAct format, we apply a lightweight system-prompt adapter that exposes the structured tool-call schema, after which checkpoints emit executable structured tool calls and enter the alternating action–observation loop. We do not exclude any baseline from the main tables.

Judge-based answer evaluation. For benchmarks evaluated by semantic equivalence rather than exact match, we use Kimi-K2.5 as the LLM-based evaluator (prompt in Appendix F, Table 7; human spot-check in Appendix G).

Real-search executor. Except for the synthetic columns of Table 4, all reported results are obtained with a real-search executor rather than the synthetic backend, and every method is run through the same executor. The `search` action is served by a commercial search API (SearchCans /api/search)

Item	Value
Environment backend	Qwen3-235B-A22B-Instruct
Decoding temperature	0
Maximum generated tokens	20480
Search result count N	5
Browse truncation length	4096
Context-linking policy	context-linked browsing

Table 10: **Synthetic environment settings.** Main settings of the shared synthetic search–browse environment.

that forwards queries to live Google and returns ranked title–URL–snippet lists in the same schema as the synthetic environment. Requests carry no explicit locale, country, or language parameter and no date filter, so the provider infers the query language from the query string itself. The number of returned results per call matches the synthetic setting.

The `browse` action fetches the requested URL with a headless Chromium browser (Playwright) and extracts main-body text with Trafilatura; when extraction yields empty or near-empty content, we fall back to a reader API and, failing that, to the raw page text. Extracted text is truncated to the same character budget as the synthetic environment, so the two environments expose observations of comparable length. Network timeouts, blocked pages, and extraction failures are surfaced to the policy as structured error observations and handled by the same recovery path described above; they do not silently terminate the episode.

All search responses are stored in a query-keyed SQLite cache, and browse responses are keyed by URL. Repeated calls within and across runs are served from this cache, which fixes exactly which documents each query returned for the duration of the evaluation. Consequently the four evaluation runs per checkpoint share one retrieval snapshot, and differences across runs reflect policy sampling rather than search-result drift. Because live web content changes over time, absolute numbers obtained from a snapshot collected at a different date may shift, while comparisons within our snapshot remain internally consistent.

E.5 Seed–Evaluation Decontamination

Our seed pool is drawn exclusively from the **official HotpotQA training split** (`hotpot_medium_bridge`). All evaluation benchmarks (HotpotQA dev, 2WikiMultiHopQA dev, Bamboogle, MuSiQue dev, GAIA, WebWalker, `xbench`, `BrowseComp-zh`) use

disjoint splits or independently constructed test sets, ensuring no overlap with our seeds by construction.

As a secondary audit, we performed exact and normalized string matching on (question, answer) pairs between the validated synthesized tasks and all evaluation sets. The audit found **zero exact-match overlaps** on questions and zero overlaps on (question, answer) pairs across all eight benchmarks. We therefore conclude that the reported gains cannot be attributed to test-time leakage from the seed source.

E.6 Pipeline Cost Comparison

Our synthetic environment uses **Qwen3-235B-A22B-Instruct** to simulate *both* the search and browse tools, so it incurs only LLM-token cost. We compare its per-call and per-step cost with two widely used real-web search APIs, Google Custom Search API and SerpAPI. All figures are based on public list prices: Qwen3-235B-A22B-Instruct on OpenRouter at \$0.071 / 1M input tokens and \$0.10 / 1M output tokens; Google Custom Search at \$5 / 1,000 queries; SerpAPI pay-as-you-go at roughly \$0.01 / query. Our environment is fully cacheable across runs and is not subject to API quotas or temporal drift; in addition, both Google CSE and SerpAPI return only search snippets, so any browse-time evidence under real-web baselines requires a separate scraping or reader service, whose cost is *not* included in Table 11.

F Prompt Templates

This appendix documents the full prompt templates used in the environment-aligned task synthesis pipeline (Section F.1) and the inference-time search agent (Section F.2).

F.1 Task Synthesis Agent Prompts

Table 12 gives an overview of all agents and their roles. The complete prompts are listed below.

Item (USD)	Ours (Qwen3-235B sim.)	Google CSE	SerpAPI
Search call (avg.)	2.5×10^{-4}	5.0×10^{-3}	1.0×10^{-2}
Browse call (avg.)	5.0×10^{-4}	—	—
Synthesis per task	0.025	—	—
Per GRPO step [†]	~2.5	~12.8	~25.6
Rate limit	none	strict quota	quota
Reproducibility	cacheable	content drift	content drift

Table 11: **Pipeline cost comparison** (USD), based on public list prices. Per-call costs of our environment are estimated from average per-call token usage under Qwen3-235B-A22B-Instruct. [†]Per-GRPO-step cost assumes 64 questions $\times$ 8 rollouts $\times$ ~ 5 search calls each; numbers for Google CSE and SerpAPI cover search calls only and exclude browse-side scraping or reader cost. Our synthetic environment is roughly 5 $\times$ cheaper than Google CSE and 10 $\times$ cheaper than SerpAPI, while remaining free of external rate limits and temporal drift.

Strategist Agent Prompt
System Prompt You are a strategic decision-making agent for multi-hop question generation. Your task is to analyze the current state, diagnose problems, and decide the next action. Goal: Generate challenging multi-hop questions that require multiple search steps (4+ guideline). Available Actions: EXPAND (add hop), REFINE (increase difficulty without new hops), ROLLBACK (undo last change), FINALIZE (output result). Decision Factors: Solver Performance, Question Quality, Difficulty Balance, Action History. User Prompt (Template) Original Question: {original_question} Current Question: {current_question} Answer: {answer} Current Level: {current_level}/{max_levels} Iteration: {iteration_count} Action History: {action_history} Latest Difficulty Report: {difficulty_report} Solver Trajectory Summary: {solver_summary} Output JSON: {"decision": "EXPAND" "REFINE" "ROLLBACK" "FINALIZE", "problem_diagnosis": ..., "root_cause": ..., "suggested_fix": ..., "expand_target": ..., "confidence": ..., "reasoning": ...}

Figure 2: **Strategist Agent prompt.** The Strategist analyzes the current state and decides the next meta-action (EXPAND / REFINE / ROLLBACK / FINALIZE).

Agent	Role	Prompt Ref.
Strategist	Meta-action selection	Figure 2
Expand	Bridge-based hop addition	Figure 3
Refine	Difficulty enhancement	Figure 4
Judge	Trajectory evaluation	Figure 5
Quality Gate	Pre-verification filtering	Figure 6
Answer Checker	Semantic equivalence	Figure 7

Table 12: Overview of synthesis agents and their prompt references.

F.2 Inference-Time Search Agent Prompt

At inference time, the trained policy interacts with the environment using the following prompt structure (Figure 8).



Figure 3: **Expand Agent prompts.** Three-step process: select target entity, select bridge from search results, rewrite question.

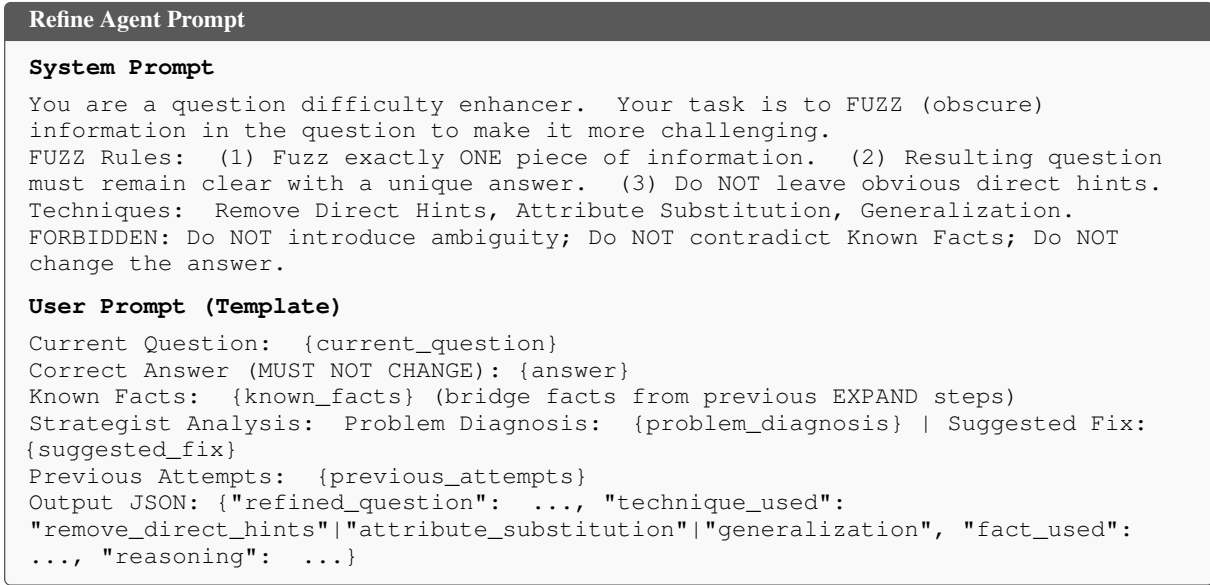


Figure 4: **Refine Agent prompt.** Increases difficulty by obscuring one piece of information without adding new hops.

Judge Agent Prompt

System Prompt

You are an expert evaluator for multi-hop question generation quality. Analyze a solver agent's attempt and provide a concise assessment.

Action Definitions: EXPAND (too easy, add hop), REFINE (shortcuts/leakage, improve wording), ROLLBACK (broken/unsolvable), FINALIZE (correct and good difficulty).

Difficulty Guidance: too_easy (0-3 steps or obvious shortcut), good (4+ steps with coherent path), too_hard (many steps but incorrect), broken (logical/factual issues).

User Prompt (Template)

Original Question: {original_question} | Current Question: {current_question}

Ground Truth Answer: {ground_truth}

Solver Trajectory: {solver_trajectory} (full trace, truncated to 15K chars)

Solver Final Answer: {final_answer}

Output JSON: {"is_correct": bool, "total_steps": int, "difficulty_level": "too_easy"|"good"|"too_hard"|"broken", "has_shortcut": bool, "recommended_action": "EXPAND"|"REFINE"|"ROLLBACK"|"FINALIZE", "reason": ..., "suggestions": [...]}

Figure 5: **Judge Agent prompt.** Evaluates solver trajectories and recommends the next meta-action.

Quality Gate Prompt

System Prompt

You are a question quality gate for multi-hop question generation. Check whether the question should proceed to solver verification.

Checks: (1) Uniqueness: unique, unambiguous answer. (2) Pseudo multi-hop: detect structural shortcuts. IMPORTANT: Ignore whether a clue is "well-known"; do NOT reject just because it is famous. Only mark shortcut if question text explicitly contains target/answer. (3) Leakage: must not directly reveal the answer.

User Prompt (Template)

Original Question: {original_question} | Current Question: {current_question}

Gold Answer: {answer} | Bridge Context: {bridge_info}

Output JSON: {"pass": bool, "failed_checks": [...], "uniqueness":

{"is_unique": bool, ...}, "pseudo_multihop": {"has_shortcut": bool, ...},

"leakage": {"leaks_answer": bool, ...}, "rollback_reason": ...}

Figure 6: **Quality Gate prompt.** Pre-verification check for uniqueness, pseudo multi-hop, and answer leakage.

Answer Checker (Semantic Equivalence) Prompt

System Prompt

You will be given a question, a solution and a ground truth answer. Your task is to determine if the answer that the solution gives is equal to the ground truth answer, ignoring minor differences in formatting, spacing, or notation. You must respond in JSON format with a 'equivalent' field that is either true or false.

User Prompt (Template)

Question: {question}

Solution: {predicted_answer}

Ground truth answer: {ground_truth}

Is the answer produced by the solution semantically equivalent to the ground-truth answer? Respond in JSON format with only the 'equivalent' field. Be strict and accurate.

Example response: {"equivalent": true}

Figure 7: **Answer Checker (Semantic Equivalence) prompt.** Used during synthesis validation and reward computation.

Inference-Time Search Agent Prompt

System Prompt

You are a helpful assistant designed to solve tasks using the provided tools. Your task is to answer the user's question by calling the appropriate tools.

1. You MUST provide your brief reasoning process before using any tool.
2. You MUST call at least one tool before giving a final answer, even if you think you know the answer.
3. Tool calls must be wrapped in `<tool_call>...</tool_call>` and contain only JSON.
4. Final answers must be wrapped in `<answer>...</answer>`.

Tools: [{"type": "function", "function": {"name": "web_search", "parameters": {"query": str, "num_results": int}}}, {"type": "function", "function": {"name": "web_browse", "parameters": {"url": str, "max_chars": int}}}]

User Prompt (Template)

Task: Please answer the following question. You must use the provided tools to search or browse for information before answering.

Question: {question}

Requirements: (1) Call at least one tool before answering. (2) Do NOT answer from internal knowledge. (3) Final answer must cite evidence from tool results. (4) Call only one tool at a time with brief reasoning. (5) Once complete, output final answer in `<answer></answer>`.

Figure 8: **Inference-time search agent prompt.** The system prompt defines tool-call format and available tools; the user prompt specifies the task.

G LLM-as-Judge Robustness Analysis

Our evaluation relies on an LLM-based judge for answer equivalence (Eq. 5). To verify that this judge produces reliable scores, we conduct a human spot-check study. The judge prompt is shown in Table 7.

G.1 Human Spot-Check Protocol

We randomly sample 100 question–answer pairs from our evaluation outputs (100 distributed across the eight benchmarks). Two human annotators independently label each pair as *equivalent* or *not equivalent* to the ground-truth answer, following the same semantic-equivalence criterion used by the LLM judge. Disagreements between human annotators are resolved by discussion. We then compare the LLM judge decisions against the consensus human labels.

G.2 Human–Judge Agreement

Table 13 reports the agreement between the LLM judge and human consensus.

Metric	Value
Human inter-annotator agreement	99.0%
LLM–Human agreement	96.0%
Precision (LLM, w.r.t. Human)	97.5%
Recall (LLM, w.r.t. Human)	92.9%
F1 (LLM, w.r.t. Human)	95.1%

Table 13: **Human spot-check results.** Agreement between the LLM judge (Kimi-K2.5) and human consensus annotations on 100 randomly sampled evaluation predictions. Human labels serve as ground truth.

G.3 Error Analysis

We categorize the cases where the LLM judge disagrees with human consensus into the following types:

Error Type	Count	%
FP (judge: equiv., human: not)	1	1.0
FN (judge: not equiv., human: equiv.)	3	3.0

Table 14: **Error categorization.** FP errors are mainly partial-date matches (e.g., “1965” vs. “1965-08-09”); FN errors arise from synonyms or abbreviations that humans accept but the judge rejects.

G.4 Qualitative Examples

Table 15 shows representative cases from the spot-check, including both agreements and disagree-

ments, illustrating the judge’s behavior across different answer formats.

H Fine-Grained Reward Ablation

Table 7 in the main text compares answer-only training with the full composite reward. Here we provide a fine-grained ablation that removes each auxiliary reward component individually to isolate its contribution.

H.1 Experimental Design

Recall the composite reward (Eq. 4):

$$R(\tau, x) = \sum_{k \in \{\text{acc}, \text{entity}, \text{uniq}, \text{decomp}\}} \lambda_k R_k(\tau, x),$$

with default weights (0.5, 0.3, 0.1, 0.1). For each ablation variant, we disable one auxiliary component (setting its weight to zero) and train under otherwise identical settings: same training-task subset, backbone (Qwen2.5-7B-Instruct), and hyperparameters. This isolates the behavioral and accuracy impact of each reward signal.

H.2 Results and Analysis

Table 16 reports the results. Each auxiliary reward component addresses a distinct failure mode; removing any one of them degrades the full system, but through qualitatively different mechanisms.

Query uniqueness (R_{uniq}) is the most critical auxiliary signal. Removing R_{uniq} causes the most severe collapse: overall accuracy drops from 44.61 to 20.01, with the deep-search average falling to just 4.45. The PerQDup ratio rises sharply from 7.18 to 16.17, indicating that the model falls into a repetitive querying loop—issuing the same or near-identical queries without making progress. Although AvgSearch is the highest among all variants (5.92), these additional search steps are largely redundant. This suggests that without an explicit incentive to diversify queries, the model degenerates into a perseverative search pattern that wastes its interaction budget on duplicate retrievals.

Original-question decomposition (R_{decomp}) prevents lazy search. Removing R_{decomp} leads to the second-largest accuracy drop (44.61 $\rightarrow$ 33.36). The most telling signal is the OrigQ metric: it surges from 1.02 to 30.95, meaning nearly one-third of all queries are verbatim copies of the original question. Without this penalty, the model defaults to a shortcut strategy: submitting the orig-

Question (abbrv.)	Predicted	Ground Truth	Judge	Human	Note
HKCU 2025 QS rank improvement over 2020?	10	10	✓	✓	Exact match; trivial case.
Restaurant near a theatre off Metro Line 13?	Dongzhimen Shuan Rou (Dongzhimen branch)	Dongzhimen Shuan Rou	✓	✓	Extra branch-name suffix; core entity matches.
Go player Zhan Ying’s 2024 record?	4W–12L	0W–2L	×	×	Clearly wrong; both agree not equivalent.
Country not in WWI, when founded?	1965	1965-08-09	✓	×	FP : Judge accepts year-only; humans require full date.
Ticket text of a porcelain palace attraction?	Porcelain Palace	Thousand-Year Porcelain Palace, Eternal Treasure	×	✓	FN : Humans accept abbreviation; judge rejects.

Table 15: **Qualitative examples from the human spot-check.** The “Judge” and “Human” columns show binary equivalence decisions. The last two rows illustrate the two disagreement types (false positive and false negative).

Variant	Acc	Ent	Uniq	Dec	MH Avg.	DS Avg.	Avg.	AvgSearch	AvgBrowse	PerQDup↓	OrigQ↓
Answer Only	✓	—	—	—	50.98	29.85	40.41	2.13	0.41	8.98	43.99
w/o Entity	✓	—	✓	✓	50.98	31.73	41.35	3.18	1.36	2.82	2.40
w/o Uniqueness	✓	✓	—	✓	35.58	4.45	20.01	5.92	1.25	16.17	1.25
w/o Decomp	✓	✓	✓	—	39.73	27.00	33.36	3.53	1.27	7.32	30.95
Full	✓	✓	✓	✓	55.90	33.33	44.61	4.00	1.70	7.18	1.02

Table 16: **Fine-grained reward ablation on Qwen2.5-7B.** ✓ indicates that the component is enabled; “—” indicates that it is removed.

inal multi-hop question directly to the search engine rather than decomposing it into targeted subqueries. This fundamentally undermines the multi-step reasoning process that deep-search tasks require.

Entity coverage (R_{entity}) has the smallest but still meaningful impact. Removing R_{entity} causes the mildest degradation: accuracy drops modestly from 44.61 to 41.35. Interestingly, PerQDup drops to its lowest value (2.82) and OrigQ remains low (2.40), suggesting that the remaining uniqueness and decomposition rewards already enforce reasonable query diversity. However, the gap to the full reward (3.26 points) is concentrated on multi-hop QA benchmarks (MH Avg. 50.98 vs. 55.90), where bridge-entity traversal is most important. Browse usage also decreases slightly (1.36 vs. 1.70), consistent with the hypothesis that entity coverage encourages page-level inspection of intermediate entities.

Summary. The three auxiliary rewards target complementary failure modes: R_{uniq} prevents degenerate repetition, R_{decomp} prevents lazy verbatim querying, and R_{entity} encourages thorough entity traversal. Their combined effect is super-

additive: no single-component ablation recovers the full reward’s performance, confirming that shaping the search *process* requires addressing multiple behavioral dimensions simultaneously.

I Qualitative Case Studies

We present four representative interaction trajectories comparing SYNSEARCH with baselines. Each case is shown as side-by-side panels—baseline on the left (red tint) and SYNSEARCH on the right (blue tint)—to highlight differences in search strategy. Both agents receive the same system prompt and tools (`web_search`, `web_browse`). Tool responses are lightly abbreviated, retaining all snippets that influenced the agent’s decisions.

J Artifact Licenses and Intended Use

We use publicly available benchmarks and pre-trained models under their respective licenses or terms of use. Specifically, the evaluation benchmarks (HotpotQA, 2WikiMultihopQA, Bamboogle, MuSiQue, GAIA, WebWalkerQA, xbench, BrowseComp-zh) are accessed through their official releases and used in accordance with their stated terms. The backbone language models

Baseline T1 Think <i>"I need to find relevant information about Suzanne Farrington's mother and her place of death."</i> Search <code>web_search("Suzanne Farrington's mother place of death")</code> Result [1] "Suzanne Farrington, Vivien Leigh's only child. She passed away presumably at home in Lower Zeals, Wiltshire..." [5] "Suzanne Farrington was the only child of film star Vivien Leigh." ⇒ <i>"Suzanne Farrington passed away in Lower Zeals. This appears to be the place of death of Suzanne Farrington's mother."</i> Answer: Lower Zeals ✕	SYNSEARCH T1 Think <i>"I need to first gather information about Suzanne Farrington and her mother."</i> Search <code>web_search("Suzanne Farrington's mother's place of death")</code> Result [1] "Suzanne Farrington, Vivien Leigh's only child. She passed away... in Lower Zeals..." [5] "...only child of film star Vivien Leigh." T2 Think <i>"Mother was Vivien Leigh. The result mentions Suzanne passed away in Lower Zeals—however, this refers to Suzanne's death, not her mother's. I need to search for Vivien Leigh's death."</i> Search <code>web_search("Vivien Leigh's place of death")</code> Result [2] "Death: 8 Jul 1967 (aged 53). Belgravia, City of Westminster, Greater London." [3] "Found dead... in the fashionable Belgravia district." T3 Think <i>"I will visit the Wikipedia page for Vivien Leigh to verify her death location."</i> Browse <code>web_browse("en.wikipedia.org/wiki/Vivien_Leigh")</code> Result "Vivien Leigh... was found dead in her London apartment..." T4 Think <i>"Wikipedia confirms London. Searching for the exact address."</i> Search <code>web_search("Vivien Leigh place of death")</code> Result [1] "8 July 1967... London, England." [2] "The death at 54 Eaton Square in Belgravia." ⇒ <i>"Vivien Leigh died at 54 Eaton Square, Belgravia, London."</i> Answer: London (Belgravia) ✓
--	--

Table 17: **Case 1 (2WikiMultiHopQA)**. Where was the place of death of Suzanne Farrington’s mother? Ground Truth: **London**. The baseline confuses Suzanne’s death with her mother’s after one search; SYNSEARCH identifies the bridge entity (Vivien Leigh) and verifies via multi-step search and browsing.

(Qwen2.5, Llama3.1, Qwen3) are used under the licenses provided by their respective developers.

The SYNSEARCH source code and synthetic task data are intended solely for academic research.

Baseline T1–T40 Think The baseline repeatedly queries “1928 Summer Olympics athletes by country” and browses Wikipedia, Olympics.com, and other pages across 20 searches and 19 browses (40 turns total). Despite extensive effort, it fails to locate structured per-country athlete counts and incorrectly identifies Malta as having the fewest athletes. Answer: MLT ×	SYNSEARCH T1 Think “I need data on athlete counts per country at the 1928 Olympics.” Search web_search(“1928 Summer Olympics athletes per country”) Result [1] Wikipedia: “706 athletes from 40 nations competed.” [3] olympedia.org: “NOCs at the 1928 Amsterdam.” T2 Think “olympedia.org seems promising for structured data.” Search web_search(“1928 Summer Olympics athletes per country list”) Result [1] olympics.com: general overview. [3] olympedia.org/editions/9: “NOCs...” T3 Think “I’ll browse olympedia.org directly for the structured table.” Browse web_browse(“olympedia.org/editions/9/countries”) Result Structured table with athlete counts: CUB: 1M/0F = 1 total; HAI: 2M/0F = 2; BUL: 5M/0F = 5; MLT: 9M/0F = 9; ... ⇒ “CUB (Cuba) has the smallest total: 1 athlete.” Answer: CUB ✓
---	---

Table 18: **Case 2 (GAIA)**. What country had the least number of athletes at the 1928 Summer Olympics? If there’s a tie, return the first alphabetically. Give the IOC country code. Ground Truth: **CUB**. The baseline exhausts 20 searches and 19 browses over 40 turns yet answers incorrectly; SYNSEARCH locates the answer in just 2 searches and 1 targeted browse. Note that the baseline panel is an uncapped diagnostic run: under the 15-step budget used for all quantitative results (Appendix E) it never emits an answer and instead loops on near-duplicate queries, so we lifted the cap for this panel only in order to show where the trajectory ends up. Step counts in this panel are therefore not comparable to the capped numbers reported in the main tables.

Baseline T1 Think "I need Zhan Ying's 2024 official match record." Search web_search(same query) Result [1] "Zhan Ying's 2022 record: 4W-12L." [3] Sina Sports: "Zhan Ying's record this year was only 4W-12L." ⇒ Accepts "4W-12L" without verifying which year "this year" refers to. Answer: 4W-12L ×	SYNSEARCH T1 Think "I need Zhan Ying's 2024 official match record." Search web_search("Zhan Ying 2024 Go match record") Result [1] "Zhan Ying competed in two official Go matches in 2024, both losses. May 13 lost to Ding Kewen; Sep 21 lost to Chen Xinyang." [3] Sina Sports: "...record this year was only 4W-12L." T2 Think "Conflicting data: 0W-2L vs. 4W-12L. I need to verify the Sina article to check which year 'this year' refers to." Browse web_browse("unifuncs.com/s/DPZ64pIz") Result Aggregated analysis page confirming two 2024 losses; also citing Sina Sports article mentioning "4W-12L" without clear year attribution. T3 Think "The '4W-12L' figure needs source verification. I'll read the original Sina article." Browse web_browse("sports.sina.com.cn/go/2024-11-27/doc-incxparp3742867.shtml") Result Article text: "In 2022, affected by the pandemic, Go events were sharply reduced; Zhan Ying's record that year was only 4W-12L." ⇒ "The 4W-12L figure refers to 2022, not 2024. For 2024 official matches: 0W-2L." Answer: 0W-2L ✓
--	--

Table 19: **Case 3 (xbench)**. What is Go player Zhan Ying’s win-loss record in official matches in 2024? Ground Truth: **0 wins, 2 losses**. The baseline accepts an unverified figure (4W-12L) that conflates years; SYNSEARCH uses web_browse to trace the source article and correctly disambiguates the time period.

Baseline	SYNSEARCH
T1 Think “I will search for the two papers.” Search <code>web_search(verbatim question)</code> Result Results include a HuggingFace dataset link and a snippet: “...first authors are Ding Jiangang...” (name truncated). ⇒ <i>Directly outputs the incomplete snippet.</i> Answer: Ding Jiangang × (Never calls <code>web_browse</code>; second author’s full name missing.)	T1 Think “I need to find both papers on the university website.” Search <code>web_search(“Chang’ an Univ. Info. Eng. + ESWA”)</code> Result [1] it.chd.edu.cn/info/1053/1247: sea-surface detection. [2] it.chd.edu.cn/info/1053/1262: lane-changing. T2 Think “Found two relevant university news pages. Browsing the first one.” Browse <code>web_browse(“it.chd.edu.cn/info/1053/1247.htm”)</code> Result Full page (7521 chars, not truncated): “...first author Ding Jiangang, advisor Prof. Li Wei...Sw-YoloX: An anchor-free detector based transformer for sea surface object detection.” T3 Think “First author confirmed: Ding Jiangang. Now browsing the second page.” Browse <code>web_browse(“it.chd.edu.cn/info/1053/1262.htm”)</code> Result Full page (6834 chars, not truncated): “...teacher Wang Zhen published as first author... A Dynamic Cooperative Lane-changing Model for Connected and Autonomous Vehicles.” ⇒ “The two first authors are Ding Jiangang and Wang Zhen.” Answer: Ding Jiangang and Wang Zhen ✓

Table 20: **Case 4 (WebWalker)**. Who are the first authors of two papers published by the School of Information Engineering at Chang’an University in Expert Systems with Applications, on sea-surface object detection and cooperative lane-changing, respectively? Ground Truth: **Ding Jiangang and Wang Zhen**. The baseline searches once with the verbatim question, obtains only a partial answer, and never browses; SYNSEARCH locates and browses the university’s news pages to extract both authors precisely.