DeepSearch-World: Self-Distillation for Deep Search Agents in a Verifiable Environment

Anonymous ACL submission

Abstract

Training tool-use agents to improve from their own experience remains challenging, as supervised fine-tuning relies on fixed teacher-distilled trajectories, while sparse-reward reinforcement learning provides weak supervision for long-horizon interactions. We present DeepSearch-Evolve, a self-distillation framework for web agents built on DeepSearch-World, a deterministic and verifiable environment with *reproducible* search and page-reading tools. DeepSearch-World contains 420K multi-hop QA tasks constructed from entity-level random walks and supports key agentic cognitive behaviors useful for self-evolving, including progress verification, grounded reflection, and failure recovery. DeepSearch-Evolve iteratively performs trajectory generation, filtering, data mixing, and fine-tuning to train stronger agents. Without distillation from more capable models, DeepSearch-World-9B achieves competitive performance compared with open-source agents, reaching 31.2% on BrowseComp, 61.5% on GAIA, and 93.4% on HotpotQA, showing that verifiable environments enable scalable self-evolution for long-horizon web agents. We will release the environment, 420K training pool, validation set, model, and code to facilitate future research on self-improving deep search agents.

1 Introduction

Recent advances in large language models (LLMs) have enabled agents that move beyond passive text generation to plan multi-step tasks, formulate search queries, read documents, browse the web, and refine answers through tool use and iterative reasoning (Schick et al., 2024; OpenAI, 2025a; Google, 2024). However, enabling these agents to improve autonomously from their own interactions remains a key challenge toward scalable self-evolving agents (He et al., 2025).

A common recipe is supervised fine-tuning (SFT) on positive trajectories distilled from the backbone model itself (Zelikman et al., 2022; Zeng et al., 2024). However, the performance easily saturates after a few optimization steps, bounded by the inherent capability of the backbone model itself and the diversity of the self-collected trajectories (Song et al., 2025b). Another way to instantiate self-evolving is to optimize tool-use agents on their own rollouts with verifiable rewards, typically using RL-style objectives such as GRPO (Guo et al., 2025a; Team et al., 2025a,c). However, these rewards are typically sparse and trajectory-level, offering little guidance on whether failures arise from query formulation, tool selection, evidence extraction, or answer synthesis (Liu et al., 2025c; Li et al., 2026).

On-policy self-distillation (OPSD) alleviates reward sparsity by matching the student's own rollouts to dense token-level distributions from a fine-grained teacher policy (Hübotter et al., 2026; Ye et al., 2026). However, in agentic tool use, such fine-grained supervision lies in actions, including tool selection, evidence verification, search query reformulation, and progress tracking (Gritta et al., 2026; Liu et al., 2025b), which requires a deterministic and verifiable interaction environment. Otherwise, the teacher distribution at each tool-use step can be noisy, limiting the direct applicability of OPSD to long-horizon agents.

In summary, as shown in Fig. 1, existing self-evolving pipelines for tool-use agents are limited by static imitation signals, sparse outcome rewards, or unreliable dense supervision. **Long-horizon agents instead require a verifiable environment that can expose process-level supervision over intermediate tool-use decisions,** Therefore, we introduce DeepSearch-World, a deterministic and verifiable environment for deep search agents. Built on DeepSearch-World, we further develop DeepSearch-Evolve, a self-distillation

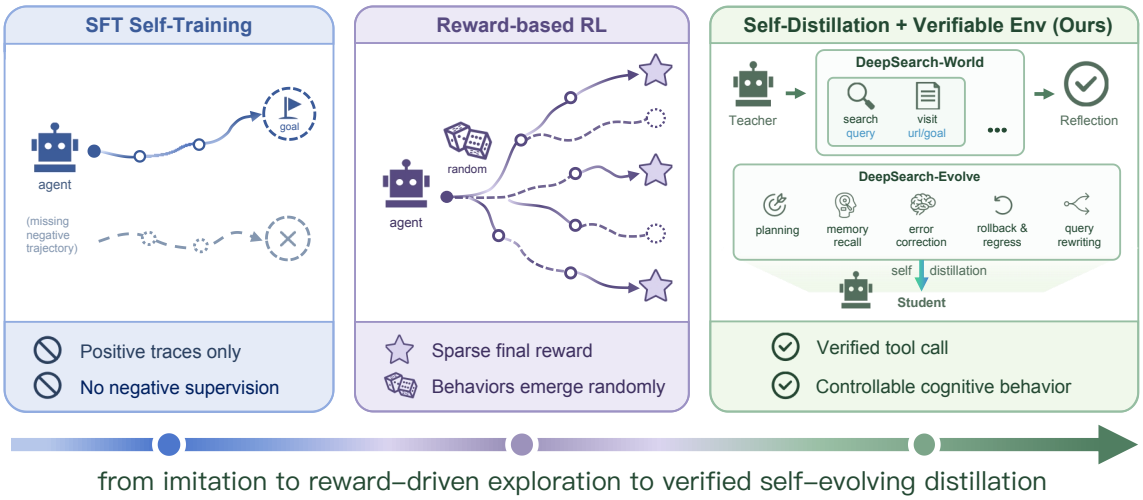

Figure 1: **Conceptual comparison of self-evolving paradigms for tool-use agents.** SFT imitates positive traces, reward-based RL learns from sparse outcomes, whereas our DeepSearch-Evolve distills verified tool-use behaviors in a deterministic environment DeepSearch-World, providing controllable process-level supervision for planning, memory, error correction, rollback, and query rewriting.

framework that enables agents to iteratively improve from their own verified tool-use experience. This work makes three contributions:

First, we introduce DeepSearch-World, a deterministic and verifiable offline environment for deep-search agents with search and browse tools, accompanied by 420K aligned multi-hop QA tasks over Wikipedia. Unlike live web environments, DeepSearch-World provides reproducible observations and entity-level verification for intermediate tool-use progress.

Second, we propose scaffold process supervision for long-horizon tool use. The teacher explicitly tracks progress, evidence, failed attempts, and recovery during rollout, and we convert these scaffold trajectories into standard ReAct-format supervision for student learning, injecting planning, memory tracking, grounded reflection, and failure recovery into student.

Third, we develop an evolving-SFT framework where agents iteratively generates trajectories, receive verified process signals, and iteratively absorb successful behaviors. This enables open-source agents to improve from verified experience without relying on synthetic trajectories from stronger proprietary models.

Experimental results show that DeepSearch-World-9B achieves competitive performance among open-source agents without distillation of strong proprietary models. Additional analyses show that DeepSearch-World-9B sustains longer tool-use interactions, uses tools more effectively, and benefits from both verified trajectory filtering and scaffold-to-ReAct conversion.

2 Related Work

Tool-Use Agents and Environments Tool-augmented LLM agents extend model capabilities through external search, API invocation, and environment interaction (Schick et al., 2024; Yao et al., 2023). Prior work explores large-scale tool-use training, web navigation agents, and trajectory-level agent tuning (Qin et al., 2023; Li et al., 2025a; Geng et al., 2026). To improve reproducibility and scalability, recent studies investigate virtual tool environments through API caching, programmatic simulators, or LLM-based environment modeling (Guo et al., 2024, 2025b; Li et al., 2025b). However, these approaches suffer from limited realism, hallucinated observations, or high serving cost. In contrast, our approach constructs a deterministic offline environment grounded in Wikipedia, enabling scalable and reproducible agent trajectory generation.

Self-evolving Agents Self-improvement methods iteratively refine model behavior using model-generated trajectories and verification signals (Zelikman et al., 2022; Singh et al., 2024; Yuan et al., 2023). GRPO-style group RL has become the dominant post-training paradigm for tool-use agents (Team et al., 2025c; Geng et al., 2025), but its reliance on sparse rewards and weak verifiability makes optimization unstable. Recent

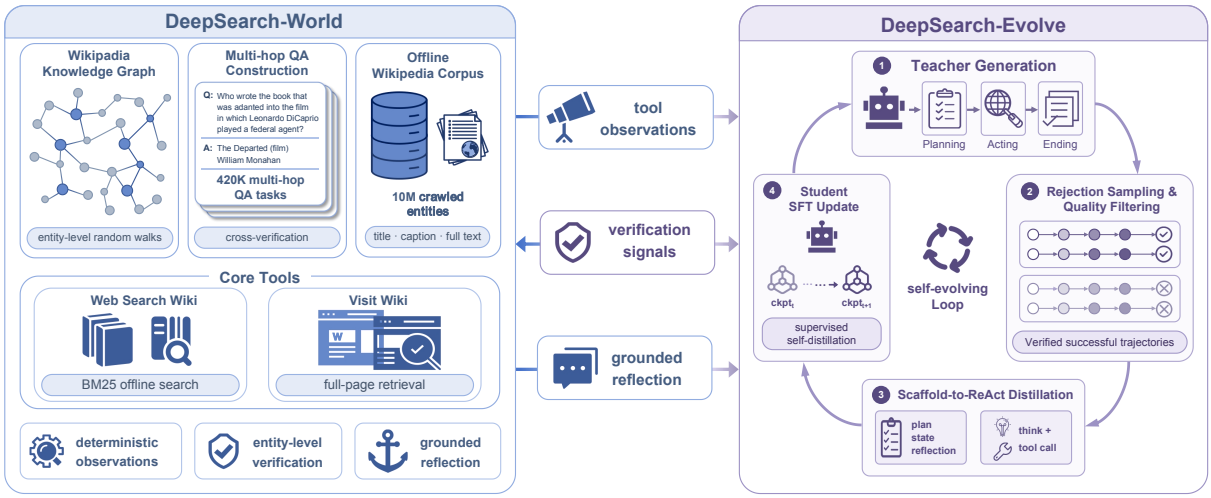

Figure 2: **Overview of the DeepSearch-World and DeepSearch-Evolve.** DeepSearch-World provides a deterministic and verifiable tool environment, while DeepSearch-Evolve performs self-distillation through a self-evolving loop of trajectory generation, rejection sampling, importance-sampling-based data mixing, and SFT training.

OPSD further seek to alleviate this distilling from on-policy generation (Hübotter et al., 2026; Zhao et al., 2026; Ye et al., 2026), but the teacher policy does not always provide a reliable or stationary target distribution. We address this by introducing an evolving SFT method and a verifiable tool environment that enables stable self-distillation.

3 Methodology

As shown in Fig. 2, we present **DeepSearch-Evolve**, a self-distillation framework for search agents consisting of three key components: (1) a verifiable tool environment **DeepSearch-World** and multi-hop QA data construction, (2) a scaffold teacher agent that generates high-quality trajectories, and (3) an iterative self-evolving training loop with asynchronous generation and training.

3.1 Verifiable Tool Environment

3.1.1 Data and Environment Construction

To support scalable self-distillation, we build a deterministic offline Wikipedia environment **DeepSearch-World** together with a large-scale multi-hop QA dataset. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the Wikipedia hyperlink graph. Following (Geng et al., 2025), we sample entity-level random walks $\tau = (v_1, \ldots, v_H)$ from $\mathcal{G}$, where each node corresponds to a target entity and the walk defines an H-hop reasoning chain. We construct 420K QA instances by obfuscating explicit entity mentions, requiring the agent to recover the target entities through reasoning, search-query revision, and evidence discovery. A large task pool broadens the induced trajectory distribution, covering diverse reasoning paths, failure modes, and recovery patterns. We analyze this effect in Sec. 4.3.1.

We then ground the QA tasks in an offline corpus to instantiate DeepSearch-World. For each instance q_i, let $\mathcal{T}_i = \{e_{i,1}, \ldots, e_{i,H}\}$ be the target entity set defined by its random walk. We crawl the corresponding Wikipedia pages for $\bigcup_i \mathcal{T}_i$ and build a local corpus $\mathcal{C}$ of approximately 10 million entries, each consisting of a title, a caption, and full article text. This alignment ensures that the evidence required by each question is searchable and verifiable within the offline environment. Implementation details are provided in Appendix A.1.

From the 420K QA pool, we reserve a held-out validation split, **DeepSearch-Val**, for training validation and behavioral analysis. It contains 377 high-quality instances whose required evidence is covered by the offline corpus and retrievable through the provided search and visit tools, enabling reliable evaluation in the deterministic environment DeepSearch-World. These instances are excluded from trajectory generation and model training. Answers are verified by a five-expert pool, with each instance independently validated by at least three experts.

3.1.2 Offline tools

The environment exposes two offline tools that follow the standard search–read workflow of web agents while keeping all observations local, deterministic, and verifiable. Formally, let $\mathcal{I}$ denote the Lucene BM25 index built over Wikipedia

corpus $\mathcal{C}$ with Pyserini (Lin et al., 2021). The search tool is a deterministic retrieval function $\texttt{web_search_wiki}(x) \mapsto \{(t_j, s_j, u_j)\}_{j=1}^{K}$ where x is a natural-language query, and each returned candidate consists of a page title t_j, a short caption s_j, and a deterministic URL u_j. This simulates a search engine that returns snippets and links rather than full documents. The visit tool is a deterministic reading function $\texttt{visit_wiki}(u_j) \mapsto d_j$ which maps a URL to the corresponding full article $d_j \in \mathcal{C}$ through a SQLite offset index, simulating a browser visit for reading detailed evidence.

Although the backend is restricted to Wikipedia, the tool schema is aligned with real web tools: search maps queries to ranked snippets and URLs, while visit maps URLs to page content. This schema alignment allows the offline tools to be replaced by live web search and visit tools without changing the action format. Implementation details are provided in Appendix A.2.

3.1.3 Environment-grounded reflection

For each question q_i, DeepSearch-World stores a ground-truth entity set $\mathcal{T}_i = \{e_{i,1}, \ldots, e_{i,H}\}$ for process-level verification. During rollout, the environment maintains a completed set $\mathcal{S}_t \subseteq \mathcal{T}_i$. A tool response o_t is considered successful if it matches any unresolved entity in $\mathcal{T}_i \setminus \mathcal{S}_t$, after which the matched entity is added to $\mathcal{S}_{t+1}$. This order-free verification identifies objective progress after each tool call without expensive LLM judgments. Failed calls trigger staged rule-based reflection toward the next unresolved entity. The first failure receives generic revision signals, while repeated failures reveal stronger guidance such as the canonical entity name or a fuzzy description. This yields grounded "search–fail–reflect–retry" trajectories and encourages query reformulation and recovery. Reflection is used only during scaffold teacher rollout and is removed or rewritten as self-correction for student.

3.2 Scaffold Teacher Agent

Following (Fang et al., 2026), we create a scaffold teacher with three phases: PLAN, ACT, and END. The scaffold is only for trajectory generation, explicitly supervising planning, memory, error recovery, and grounded answering before distillation.

Plan. Given a question q, the teacher initializes a structured progress state s_0 for task decomposition and evidence tracking. Fig. 3 shows the state

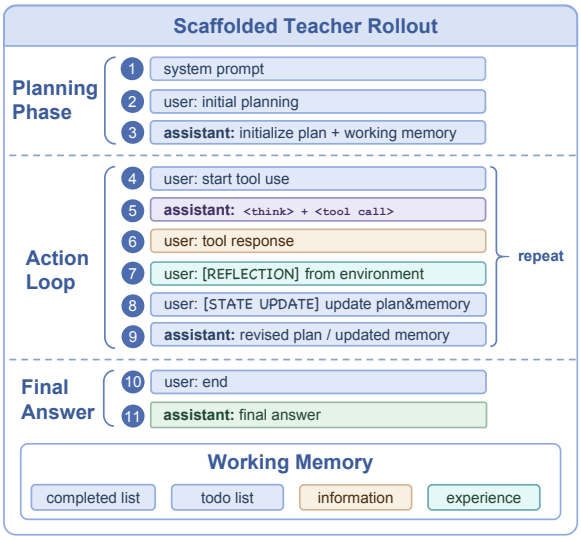

Figure 3: **Three-phase scaffolded teacher rollout.** The teacher proceeds through the initial *Plan*, iterative *Act*, and *End*

contains four writable fields: `completed_list` for confirmed subgoals, `todo_list` for following actions, `experience` for useful lessons from failed attempts, and `information` for evidence extracted from observations.

Act. The teacher then performs up to $T_{\max}$ interaction steps. At step t, it selects a tool call a_t from the current progress state s_t, receives an observation o_t, and updates

$$s_{t+1} = \mathcal{U}(s_t, a_t, o_t, r_t), \tag{1}$$

where r_t is the environment-grounded reflection returned upon retrieval failure. This process records verified evidence, failed strategies, and revised goals, yielding trajectories with both successful tool use and recoverable failures.

End. When evidence is sufficient or the budget is exhausted, the teacher enters END and generates a concise answer grounded in verified working memory, reducing unsupported synthesis and hallucination. An example rollout is shown in Fig. 4.

3.3 Self-Evolving Training Loop

DeepSearch-Evolve uses an iterative self-evolving loop that alternates between scaffold trajectory generation and training. At round R, the current model π_{θ_R} generates trajectories in the verifiable virtual-tool environment as the teacher. Then verified successful trajectories are converted into ReAct-format supervision to train the student

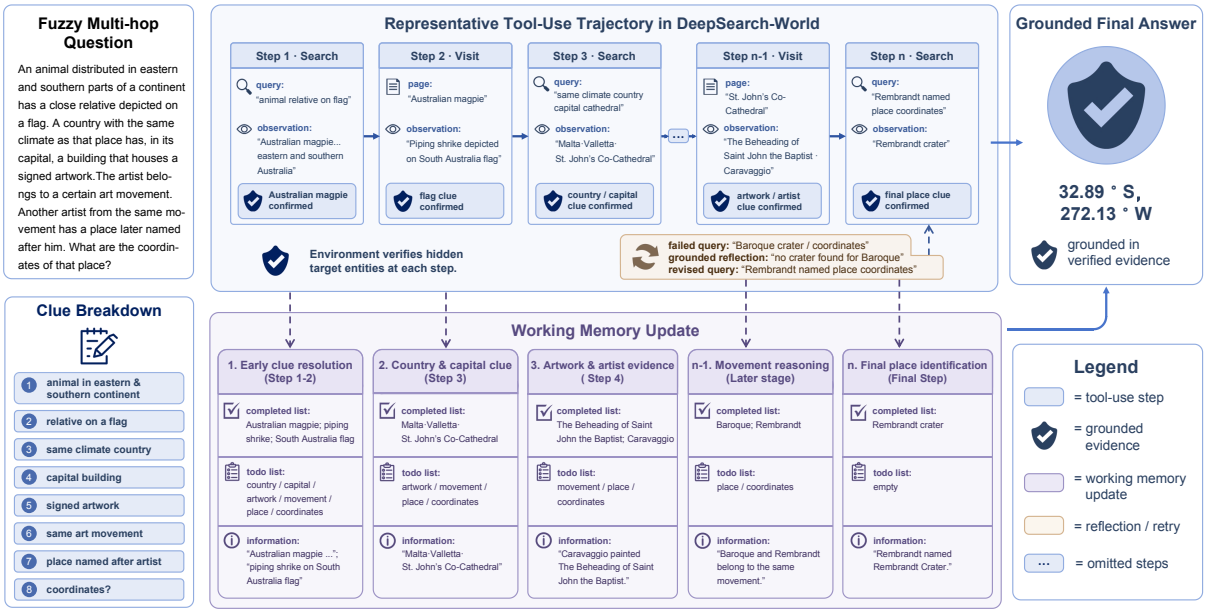

Figure 4: **Example scaffold trajectory generated by the teacher agent.** The teacher decomposes a fuzzy multi-hop question, alternates between search and visit tools, updates working memory with verified evidence, uses grounded reflection to recover from failed searches, and produces a final answer supported by verified observations.

$\pi_{\theta_{R+1}}$, which will be the next teacher. This enables the agent to improve from its own verified tool-use experience rather than relying on a fixed demonstration set.

Trajectory generation and verification. For each question q, π_{θ_R} generates a scaffold trajectory τ with the teacher agent in Sec. 3.2, including tool calls, observations, state updates, reflections, and a final answer. We retain trajectories with correct answer and further prompt Qwen3.5-9B (Qwen Team, 2026) to apply trajectory-level filtering to remove redundant evidence, weak goal alignment, and inconsistent reasoning. The prompt we use is in Appendix C.2.

Scaffold-to-ReAct conversion. The scaffold improves teacher trajectory quality but is not exposed to the student, since our goal is to train a deployable ReAct agent rather than a model dependent on external planning or reflection prompts. Each retained scaffold trajectory is therefore converted into standard ReAct-format supervision by removing stage-specific prompts and rewriting progress states and reflections into the assistant reasoning trace.

For each tool-use step t, the target <think> block is constructed as

$$\texttt{<think>}_t = P_t \oplus R_t \oplus A_t. \tag{2}$$

Here, P_t is rewritten from the current progress state s_t, summarizing completed subgoals, pending targets, failed-search experience, and verified evidence; R_t rewrites the environment reflection r_t as self-correction for query adjustment; and A_t preserves the local rationale for action a_t. The action a_t and observation o_t are kept unchanged. This conversion trains a standard ReAct policy while distilling scaffold-induced planning, memory tracking, and error recovery. An example converted trajectory is provided in Appendix D.

Supervised self-distillation update. We update the agent with SFT on environment-verified ReAct trajectories, which can be viewed as hard-label token-level policy distillation. Given a verified trajectory $\tilde{\tau} = (x, y_{1:T})$, SFT minimizes

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = \mathbb{E}_{\tilde{\tau}\sim\tilde{\mathcal{D}}(R)} \sum_{t=1}^{T} \mathrm{KL}\big(\delta_{y_t} \,\|\, \pi_\theta(\cdot \mid x, y_{<t})\big), \tag{3}$$

which is equivalent to negative log-likelihood up to a constant. Meanwhile, on-policy self-distillation (OPSD) matches the student to a soft teacher distribution on student-induced prefixes. Let $\pi_\theta^t = \pi_\theta(\cdot \mid x, \hat{y}_{<t})$ and $q^t = q(\cdot \mid x, \hat{y}_{<t})$. Then OPSD minimizes

$$\mathcal{L}_{\mathrm{OPSD}}(\theta) = \mathbb{E}_{x,\, \hat{y}_{<t}\sim\pi_\theta} \sum_{t} \mathrm{KL}\big(q^t \,\|\, \pi_\theta^t\big). \tag{4}$$

Thus, both objectives provide dense token-level supervision, but differ in prefix and target distri-

Model / Agent	BrowseComp	BrowseComp-ZH	HLE	GAIA	xbench	HotpotQA
Proprietary Agents						
OpenAI Deep Research	**51.5**	42.9	**26.6**	67.4	–	–
OpenAI-o3	49.7	**58.1**	20.2	**70.5**	**65.0**	–
Open Source Agents						
R1-Searcher	1.0	–	5.4	8.3	–	62.4
Search-R1	0.4	–	13.0	18.7	–	63.2
ZeroSearch	1.4	–	8.6	9.9	–	32.4
ASearcher	3.2	–	13.8	22.1	–	71.0
DeepResearcher	1.8	–	6.0	24.0	–	56.6
PokeeResearch	6.2	–	17.6	49.2	–	71.6
WebSailor	6.7	14.2	12.8	37.9	34.3	92.8
OffSeeker-DPO	12.8	26.6	17.6	51.5	48.0	–
WebExplorer	15.7	32.0	17.3	50.0	53.7	–
Marco-DR	31.4	47.1	–	69.9	42.0	–
MiroThinker-v1.0	31.1	40.2	21.5	66.4	34.0	–
DeepDive	6.3	15.1	–	–	35.0	–
Ours						
Qwen3.5-9B-Instruct	7.4	13.5	16.7	23.9	20.0	45.3
DEEPSEARCH-WORLD-9B	31.2	36.4	25.7	61.5	49.0	**93.4**
Δ	+23.8	+22.9	+9.0	+37.6	+29.0	+48.1

Table 1: Main results on deep search and related reasoning benchmarks. All scores are reported as percentages. A dash indicates that the corresponding result is unavailable or not applicable.

butions. SFT uses verified offline prefixes with hard targets, whereas OPSD uses on-policy prefixes with soft targets.

We adopt evolving SFT for stability in long-horizon tool use. Fully on-policy distributional supervision is costly and can be unreliable, as student rollouts may drift into low-quality tool states where teacher token distributions become noisy or misleading. Instead, we sample scaffold rollouts from the current agent, filter them through rejection sampling and filtering, and convert them into ReAct traces. These traces supervise planning, tool invocation, error recovery, and final synthesis without relying on per-token guidance over unverified states. The resulting update is partially on-policy in data generation but teacher-forced in optimization, trading full distributional matching for stable, efficient, and reusable supervision.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate on seven deep search and related reasoning benchmarks: BrowseComp (Wei et al., 2025), BrowseComp-ZH (Zhou et al., 2025), HLE (Center for AI Safety et al., 2026), GAIA (Mialon et al., 2023), xbench (Chen et al., 2025), HotPotQA (Yang et al., 2018) and Search-QA (Jin et al., 2025a). Details of these benchmarks are provided in Appendix B.1.

Baselines. We compare against proprietary and open-source models under two evaluation settings. *Proprietary Agents* includes OpenAI Deep Research (OpenAI, 2025a) and OpenAI-o3 (OpenAI, 2025b). In the *Open Source Agents* setting, we test PokeeResearch (Wan et al., 2026), WebSailor (Li et al., 2025a), Marco-DR (Zhu et al., 2026), MiroThinker-v1.0 (Team et al., 2025b), WebExplorer (Liu et al., 2025a), DeepDive (Lu et al., 2025), R1-Searcher (Song et al., 2025a), Search-R1 (Jin et al., 2025b), ZeroSearch (Sun et al., 2025), ASearcher (Gao et al., 2025), Offseeker (Zhou et al., 2026), and DeepResearcher (Zheng et al., 2025). Finally, we compare DeepSearch-World-9B with Qwen3.5-9B-Instruct backbone (Qwen Team, 2026).

Training Configuration. We train DeepSearch-World-9B from Qwen3.5-9B for 11 self-evolving rounds in DeepSearch-World. Each round generates trajectories for 10,000 instances and triggers training once 4,000 trajectories pass rejection sampling and quality filtering, with trajectory genera-

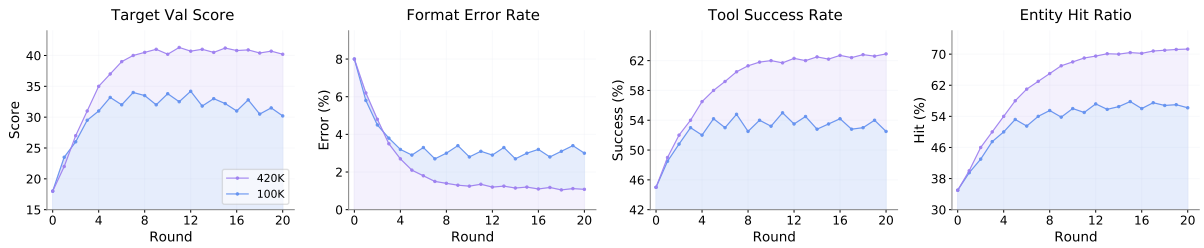

Figure 5: Data scale effect in evolving SFT. Compared with 100K total training data, 420K QA lead to a higher target-validation plateau, lower residual format errors, and more stable gains in tool success and entity hit ratio.

tion capped at $T_{\max} = 30$ steps. Since generation and training are asynchronous, we use importance sampling when new trajectories outpace SFT updates, sampling across rounds with decay factor $\gamma = 0.5$ to prioritize recent verified data while retaining earlier trajectories to mitigate catastrophic forgetting. We fine-tune for one epoch per round with Llama Factory (Zheng et al., 2024). To reduce the offline-to-real execution gap, we further apply GRPO on 1,600 real-tool instances, using Google SerpAPI (Google, 2025) for search and Jina (Jina.ai, 2025) for page retrieval. Full hyperparameters are provided in Appendix B.3.

4.2 Main Results

Tab. 1 compares DeepSearch-World-9B with proprietary systems and recent open-source deep-search agents. A key distinction is the supervision source. Many competitive open-source agents rely on stronger-model trajectories or multi-agent synthetic pipelines, e.g., O-Researcher uses multi-agent distillation, and Marco-DR synthesizes SFT trajectories from frontier foundation models. In contrast, DeepSearch-World-9B is optimized only from its own environment-verified rollouts and achieves competitive open-source performance. This suggests that deterministic environment verification can replace part of the supervision traditionally supplied by stronger teachers or external synthetic pipelines.

Under the same tool setting, DeepSearch-World-9B consistently improves over Qwen3.5-9B-Instruct across all benchmarks, with gains of +23.8 on BrowseComp, +22.9 on BrowseComp-ZH, +9.0 on HLE, +37.6 on GAIA, +29.0 on xbench, and +48.1 on HotpotQA. These gains indicate that evolving self-distillation learns transferable tool-use behaviors, including query reformulation, evidence grounding, and multi-step synthesis. The lower BrowseComp-ZH score is expected because training uses only English trajectories, though the improvement over the backbone still indicates partial cross-lingual transfer.

4.3 Analysis

4.3.1 Self-Evolving Trajectory Quality Over Rounds

Fig. 5 compares evolving SFT under different data-pool scales, with each round trained on 4K verified trajectories. The 420K pool provides sufficient unseen instances across rounds, whereas the 100K pool is exhausted much earlier, causing later trajectory generation to rely on previously rejected or low-yield instances with weaker supervision. Consequently, the 420K setting reaches a higher validation plateau, lower residual format error, and more stable gains in tool success and entity hit ratio, while the 100K setting improves early but saturates sooner and fluctuates more. This suggests that evolving SFT relies on data-pool diversity rather than repeated exposure. Duplicated trajectories risk overfitting to narrow interaction patterns, whereas a larger pool better covers world knowledge, task forms, and failure-recovery behaviors, improving generalization and preserving agentic skills across rounds. Based on this trend, we use 11 evolving rounds for the main experiments, which provides strong validation performance before the curve largely saturates.

4.3.2 Tool-Use Behavior Analysis

Fig. 6 shows a clear gap in long-horizon tool use. Qwen3.5-9B-Instruct terminates after only 4.7 rounds on average, often giving premature final answers, whereas DeepSearch-World sustains 18.0 rounds and gathers substantially more evidence for multi-step reasoning. Qwen3.5-9B-Instruct also under uses `visit` for only 0.9 calls, indicating limited evidence access and verification; in contrast, DeepSearch-World performs 5.4 `visit` calls on average, suggesting stronger document-level grounding. We further report an *Advanced*

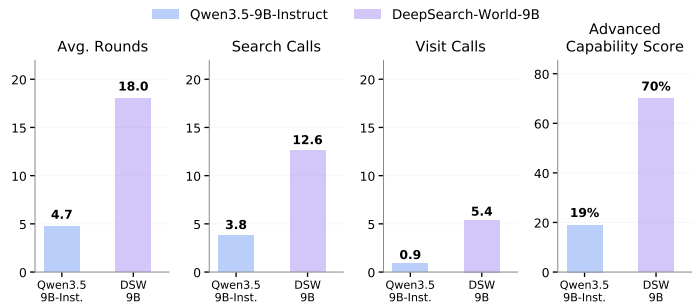

Figure 6: Tool-use behavior and advanced capability on DEEPSEARCH-VAL. Compared with Qwen3.5-9B-Instruct, DeepSearch-World sustains longer interactions, performs more `search` and `visit` calls, and achieves a substantially higher advanced capability score.

Table 2: Ablation of rejection sampling and trajectory quality filtering. **RS only** selects answer-correct trajectories, **QF only** applies the trajectory quality filter, and **w/o both** removes both. Results are percentages averaged over three runs.

Filtering	SearchQA
RS only	54.9
QF only	48.1
w/o both	46.4
SDAR	49.0
Skill-SD	47.8
DeepSearch-World-9B (**RS + QF**)	**58.2**

Configuration	DeepSearch-Val
Qwen3.5-9B-Instruct	8.5
SFT	25.0
Full pipeline	31.9
w/o state internalization	23.5
w/o reflection rewriting	16.7
w/o both	14.8

Table 3: Ablation on trajectory-to-training-data conversion strategies. Results are percentages averaged over three runs.

Capability Score, evaluated by an LLM judge over five dimensions: planning, memory maintenance, reasoning, self-correction, and evidence gathering toward the final answer, whose rubric is in Appendix B.5. Qwen3.5-9B-Instruct scores only 19%, while DeepSearch-World reaches 70%, demonstrating stronger tool-use strategy.

4.4 Ablation Studies

4.4.1 Impact of State Internalization

Tab. 3 studies the conversion of scaffolded trajectories into ReAct-format SFT data. Reflection rewriting only masks entity names in environment feedback to prevent answer leakage, while preserving the feedback structure rather than replacing it with oracle answers.

The full pipeline outperforms vanilla SFT on 10K converted trajectories, improving DeepSearch-Val from 25.0 to 31.9 , demonstrating the benefit of evolving self-distillation. Reflection rewriting is critical: removing it drops performance to 16.7 and 15.7, respectively, as raw reflections contain artifacts such as `[REFLECTION]` tokens and incomplete environment messages that distort the `<think>` distribution. State internalization provides smaller but consistent gains by injecting planning, memory, and progress tracking into the reasoning trace.

4.4.2 Impact of Trajectory Quality Filtering

Tab. 2 ablates rejection sampling (RS) and trajectory quality filtering (QF) in teacher trajectory generation. RS gives the dominant gain, raising SearchQA from 46.4 to 54.9, indicating that answer-correctness verification is the key safeguard for self-distillation. QF alone brings a smaller gain, but combined with RS in DeepSearch-World-9B, it further improves performance to 58.2 by filtering redundant, weakly aligned, or inconsistent answer-correct traces. The clear advantage over OPSD-based SDAR and Skill-SD suggests that evolving SFT with verified trajectory filtering offers a more stable and effective self-evolving path.

5 Conclusion

We introduced DeepSearch-World, a deterministic tool environment built over an offline Wikipedia corpus for deep search agents. It provides reproducible observations, entity-level progress verification, and grounded reflection signals, making tool-use trajectories stable and verifiable. Building on this environment, DeepSearch-Evolve performs self-distillation by generating verified trajectories and converting scaffold supervision into standard ReAct-format training data. This process helps the student agent acquire planning, memory maintenance, self-correction, and evidence-grounded tool use, offering a scalable path for improving tool-use agents from self-generated experience. Experiments show that DeepSearch-World-9B substantially improves over Qwen3.5-9B across deep-search and reasoning benchmarks, while remaining competitive with strong open-source agents.

Limitation

First, our current environment is built on Wikipedia, which limits coverage and domain diversity. Extending verifiable environments to broader knowledge sources could improve generalization. Second, our update rule relies on evolving SFT, while RL-style or OPSD-based updates may further improve generalization and flexibility. However, how to inject higher-level capabilities, such as planning, error recovery, and tool-use strategies, into RL training remains underexplored. We believe addressing these limitations will further broaden the scope of verifiable self-evolution for tool-use agents.

Ethical Considerations

This work uses publicly available Wikipedia content, which is licensed under CC-BY-SA, and does not involve private, sensitive, or personally identifiable information. We will release DeepSearchWorld and the constructed dataset for research purposes under a CC BY-SA-compatible license, with attribution to Wikipedia contributors.

Our environment is fully offline and deterministic, reducing risks from uncontrolled web access during training. The dataset does not collect private or newly generated personally identifiable information. It may contain publicly available names and biographical facts already present in Wikipedia. However, Wikipedia may contain societal biases, and models trained on it may inherit such biases. Future work should explore more diverse data sources and stronger safety alignment for tool-use agents.

References

Center for AI Safety, Scale AI, and HLE Contributors Consortium. 2026. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649:1139–1146.

Kaiyuan Chen, Yixin Ren, Yang Liu, Xiaobo Hu, Haotong Tian, Tianbao Xie, Fangfu Liu, Haoye Zhang, Hongzhang Liu, Yuan Gong, and 1 others. 2025. xbench: Tracking agents productivity scaling with profession-aligned real-world evaluations. *arXiv preprint arXiv:2506.13651*.

Tianqing Fang, Zhisong Zhang, Xiaoyang Wang, Rui Wang, Can Qin, Yuxuan Wan, Jun-Yu Ma, Ce Zhang, Jiaqi Chen, Xiyun Li, Yonglin Wang, Jingchen Ni, Tianshi Zheng, Chun Chen, Wenhao Yu, Zhenwen Liang, Hongming Zhang, Haitao Mi, and Dong Yu. 2026. Cognitive kernel-pro: A framework for deep research agents and agent foundation models training. *Preprint*, arXiv:2508.00414.

Jiaxuan Gao, Wei Fu, Minyang 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*.

Xinyu Geng, Peng Xia, Zhen Zhang, Xinyu Wang, Qiuchen Wang, Ruixue Ding, Chenxi Wang, Jialong Wu, Yida Zhao, Kuan Li, and 1 others. 2025. Webwatcher: Breaking new frontier of vision-language deep research agent. *arXiv preprint arXiv:2508.05748*.

Xinyu Geng, Yanjing Xiao, Yuyang Zhang, Hanwen Wang, Xinyan Liu, Rui Min, Tianqing Fang, and Yi R Fung. 2026. Geobrowse: A geolocation benchmark for agentic tool use with expert-annotated reasoning traces. *arXiv preprint arXiv:2604.04017*.

Google. 2024. Try deep research and our new experimental model in gemini, your ai assistant.

Google. 2025. Serpapi.

Milan Gritta, Debjit Paul, Xiaoguang Li, Lifeng Shang, Jun Wang, and Gerasimos Lampouras. 2026. Process evaluation for agentic systems. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 2678–2692.

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

Zhicheng Guo, Sijie Cheng, Yuchen Niu, Hao Wang, Sicheng Zhou, Wenbing Huang, and Yang Liu. 2025b. Stabletoolbench-mirrorapi: Modeling tool environments as mirrors of 7,000+ real-world apis. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 5247–5270.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shimin Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool utilization of large language models. *arXiv preprint arXiv:2403.07714*.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Hongming Zhang, Tianqing Fang, Zhenzhong Lan, and Dong Yu. 2025. OpenWebVoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27545–27564, Vienna, Austria. Association for Computational Linguistics.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and 1 others. 2026. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025a. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025b. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.

Jina.ai. 2025. Jina.

Jiazheng Li, Yawei Wang, Qiaojing Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. 2026. Salt: Step-level advantage assignment for long-horizon agents via trajectory graph. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 4709–4725.

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

Yuetai Li, Huseyin A Inan, Xiang Yue, Wei-Ning Chen, Lukas Wutschitz, Janardhan Kulkarni, Radha Poovendran, Robert Sim, and Saravan Rajmohan. 2025b. Simulating environments with reasoning models for agent training. *arXiv preprint arXiv:2511.01824*.

Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Junteng Liu, Yunji Li, Chi Zhang, Jingyang Li, Aili Chen, Ke Ji, Weiyu Cheng, Zijia Wu, Chengyu Du, Qidi Xu, and 1 others. 2025a. Webexplorer: Explore and evolve for training long-horizon web agents. *arXiv preprint arXiv:2509.06501*.

Shunyu Liu, Minghao Liu, Huichi Zhou, Zhenyu Cui, Yang Zhou, Yuhao Zhou, Jialiang Gao, Heng Zhou, Yunhao Yang, Wendong Fan, and 1 others. 2025b. Veriweb: Verifiable long-chain web benchmark for agentic information-seeking. *arXiv preprint arXiv:2508.04026*.

Xiaoqian Liu, Ke Wang, Yuchuan Wu, Fei Huang, Yongbin Li, Junge Zhang, and Jianbin Jiao. 2025c. Agentic reinforcement learning with implicit step rewards. *arXiv preprint arXiv:2509.19199*.

Rui Lu, Zhenyu Hou, Zihan Wang, Hanchen Zhang, Xiao Liu, Yujiang Li, Shi Feng, Jie Tang, and Yuxiao Dong. 2025. Deepdive: Advancing deep search agents with knowledge graphs and multi-turn rl. *arXiv preprint arXiv:2509.10446*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

OpenAI. 2025a. Deep research system card.

OpenAI. 2025b. Introducing openai o3 and o4-mini.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Qwen Team. 2026. Qwen3.5: Towards native multimodal agents.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, and 1 others. 2024. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. 2025a. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*.

Yuda Song, Hanlin Zhang, Carson Eisenach, Sham Kakade, Dean Foster, and Udaya Ghai. 2025b. Mind the gap: Examining the self-improvement capabilities of large language models. In *International Conference on Learning Representations*, volume 2025, pages 39894–39931.

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

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, and 1 others. 2025a. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

MiroMind Team, Song Bai, Lidong Bing, Carson Chen, Guanzheng Chen, Yuntao Chen, Zhe Chen, Ziyi Chen, Jifeng Dai, Xuan Dong, and 1 others. 2025b. Mirothinker: Pushing the performance boundaries of open-source research agents via model, context, and interactive scaling. *arXiv preprint arXiv:2511.11793*.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, and 1 others. 2025c. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*.

Yi Wan, Jiuqi Wang, Liam Li, Jinsong Liu, Ruihao Zhu, and Zheqing Zhu. 2026. Rethinking the design of reinforcement learning-based deep research agents. *Preprint*, arXiv:2510.15862.

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

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 *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. 2026. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. 2026. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

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, Suzhou, China. Association for Computational Linguistics.

Peilin Zhou, Bruce Leon, Xiang Ying, Can Zhang, Yifan Shao, Qichen Ye, Dading Chong, Zhiling Jin, Chenxuan Xie, Meng Cao, and 1 others. 2025. Browsecomp-zh: Benchmarking web browsing ability of large language models in chinese. *arXiv preprint arXiv:2504.19314*.

Yuhang Zhou, Kai Zheng, Qiguang Chen, Mengkang Hu, Qingfeng Sun, Can Xu, and Jingjing Chen. 2026. Offseeker: Online reinforcement learning is not all you need for deep research agents. *arXiv preprint arXiv:2601.18467*.

Bin Zhu, Qianghuai Jia, Tian Lan, Junyang Ren, Feng Gu, Feihu Jiang, Longyue Wang, Zhao Xu, and Weihua Luo. 2026. Marco deepresearch: Unlocking efficient deep research agents via verification-centric design. *arXiv preprint arXiv:2603.28376*.

A Details of Data and environment construction.

We construct a large-scale multi-hop QA dataset of 420K instances designed to require genuine multi-step tool use. The construction pipeline has two phases.

A.1 Data and Environment Construction

Starting from seed Wikipedia articles, we perform BFS-based exploration of the Wikipedia hyperlink graph:

1. **Knowledge Tree Construction.** For each seed article, we perform breadth-first exploration. At each level, an LLM (Gemini-3-Pro) selects the b most relevant hyperlinks (default b=3) from the current page. Exploration continues for up to $d_{\max}$ levels (default $d_{\max}$=4) or until $p_{\max}$ pages (default $p_{\max}$=6) have been visited. Each edge in the tree records the semantic relationship between parent and child articles.

2. **Subtree Sampling.** From the complete knowledge tree, we randomly sample 2–4 subtrees. Each subtree defines a set of entities (nodes) and their relationships (edges).

3. **Question Generation.** For each sampled subtree, the LLM generates a complex multi-hop question that requires locating and cross-referencing information about all entities in the subtree. The question is designed to be answerable only by consulting multiple Wikipedia articles.

4. **Feature Fuzzification.** To prevent memorization and ensure genuine multi-step reasoning, we apply *feature fuzzification*: specific values in the question are replaced with approximate descriptors (e.g., "2014" → "the 2010s", "Beijing, China" → "a major Chinese city"). This forces the agent to search for and verify specific facts rather than pattern-matching from training data.

The tool environment requires a complete, preprocessed Wikipedia corpus:

1. Wikipedia XML dumps are processed via WikiExtractor to obtain clean article text.

2. Extracted articles are merged into a single JSONL file with `{id, contents, caption}` schema.

3. A SQLite offset index is built for random access by article title.

4. A Pyserini BM25 index is constructed over the full corpus for keyword retrieval.

5. Missing text (articles not in the dump) is supplemented via controlled online scraping with rate limiting.

The resulting corpus contains approximately 10 million entries, providing comprehensive coverage of English Wikipedia.

A.2 Tool Implementation Details

`web_search_wiki`. `web_search_wiki` implements the search interface of the virtual Wikipedia environment. It takes a natural-language query as input and returns the top-k retrieved articles, with k=5 by default. The retrieval backend is a Lucene BM25 index constructed with Pyserini (Lin et al., 2021) over the offline Wikipedia corpus. Each indexed document contains three fields: `keyword`, `caption`, and `contents`, corresponding to the article title, lead-section summary or feature description, and full article text, respectively. For each retrieved article, the tool returns a structured result

$$r_i = \{\texttt{keyword}_i, \texttt{caption}_i, \texttt{url}_i\}, \quad i = 1, \ldots, k, \tag{5}$$

where `url` is a deterministic Wikipedia-style URL generated from the normalized article title. Thus, the agent receives the same type of information as in a real search engine interface: a ranked list of page titles, snippets, and links, rather than full documents. The BM25 index is initialized once and shared across agent workers to reduce loading overhead.

`visit_wiki`. `visit_wiki` implements the page-reading interface. It takes a Wikipedia-style URL as input, either returned by `web_search_wiki` or directly generated by the agent, and retrieves the corresponding full article from the offline corpus. To support efficient random access, we use a two-level offset-indexed storage design. First, a SQLite database maps each normalized article title to its byte offset and length in the JSONL corpus. Second, the JSONL corpus stores the full article content. Given a URL, the tool extracts and normalizes the article title, looks up its offset and length in SQLite, and then directly seeks to the corresponding byte range in the corpus file, avoiding sequential scanning.

The title normalization pipeline includes underscore-to-space conversion, URL decoding, parenthetical disambiguation handling, and Unicode normalization. To reduce I/O overhead, the SQLite index can be loaded into shared memory at startup and reused by parallel agent workers. When an article is longer than a predefined threshold, the environment optionally calls a lightweight summarization model to return a condensed page representation containing key facts and evidence; if summarization fails, the tool falls back to deterministic character-level truncation. This design preserves the input–output schema of a real browser-visit tool while ensuring deterministic and efficient offline retrieval.

Tool Definition In the ReAct framework, each tool is defined through a structured prompt that specifies both its callable format and its semantic capability. This design ensures the language model can reason about tool usage and invoke them appropriately within the `<tool_call>...</tool_call>` block during interaction. Cache is used to prevent search repeatably. In detail, our tools are defined as follows.

Tool: `Web Search`

Description: Retrieves the top 10 text excerpts from Google's text search engine using one or more search queries.
Arguments:

- `queries` (array of strings): List of search queries. *(Required)*

Tool: `Visit`

Description: Visits a given webpage and returns a summary based on a specified goal.
Arguments:

- `url` (string): The target webpage URL. *(Required)*
- `goal` (string): The goal or information the agent seeks from the webpage. *(Required)*

A.3 Teacher Rollout Implementation Details

Interaction budget and context management. During teacher rollout, the agent is allowed to interact with the virtual environment for up to $T_{\max}$ steps. To control context length, we use a sliding-window context strategy: the most recent w interaction steps are retained in full detail, while earlier steps are summarized into the progress state. This preserves recent observations and tool-use rationales while preventing long trajectories from exceeding the context window.

Text-based function calling. We implement tool use with a text-based function-calling protocol. Each tool call is generated as plain text with delimiter tokens specifying the function name, arguments, tool result, and returned observation:

```
FUNCTION: web_search_wiki
ARGS: {"query": "..."}
RESULT: [{"keyword": "...", "caption":
"..."}]
RETURN: ...
```

The delimiters are selected to be unlikely to appear in natural text and are parsed with regular expressions for tool dispatch. This protocol is backend-agnostic and allows the same rollout code to be used with models that do not support native function-calling APIs.

Observation truncation. Tool observations are truncated when necessary to fit the rollout context. For long retrieved pages, the environment either returns a condensed representation or applies deterministic truncation, depending on the configuration described in Appendix A.2.

B Experimental Details

B.1 Benchmark Details

We evaluate the agent on six search-intensive benchmarks covering English browsing, Chinese browsing, expert-level reasoning, and search-based QA. BrowseComp contains 1,266 English browsing questions designed to test whether agents can persistently locate hard-to-find and entangled information on the web. BrowseComp-ZH extends this setting to the Chinese web with 289 natively constructed multi-hop questions across 11 domains, requiring agents to handle Chinese-specific retrieval challenges such as fragmented platforms, implicit expressions, and heterogeneous information sources. HLE evaluates frontier-level academic reasoning with 2,158 English multiple-choice questions spanning humanities, science, and mathematics. GAIA-Text contains 103 text-only questions from the GAIA development split, focusing on real-world multi-step

queries that require web search and reasoning. xBench-DeepSearch contains 100 Chinese deep-search questions for evaluating web search and reasoning ability, with encrypted public data to mitigate benchmark contamination. Search-QA follows prior search-based QA evaluations and includes both single-hop retrieval datasets, such as NQ, TriviaQA, and PopQA, and multi-hop QA datasets, such as HotpotQA, 2Wiki, MuSiQue, and Bamboogle, testing whether agents can retrieve and compose evidence across one or multiple supporting documents.

B.2 Baseline Details

We compare against proprietary models, open-source backbones, and open-source search agents under two evaluation settings. In the *Direct Reasoning* setting, models answer in a single pass without external retrieval or tool use. This group includes GPT-5, Claude-4-Sonnet, Claude-3.7-Sonnet, OpenAI Deep Research, OpenAI-o3, and the Qwen3.5-9B-Instruct backbone.

In the *Open Source Agents* setting, we compare with recent search and deep-research agents at comparable model scales, including R1-Searcher-7B, Search-R1-7B, ZeroSearch-7B, ASearcher-7B, DeepResearcher-7B, PokeeResearch-7B, WebSailor-7B, WebExplorer-8B, Marco-DR-8B, MiroThinker-v1.0-8B, and DeepDive-9B. Since publicly reported deep-search agents built exactly on 9B backbones are still limited, we include nearby 7B–8B agents to provide a broader and more informative comparison within the same small-model regime. This comparison is therefore not intended as a strict parameter-matched contest, but as a practical assessment against recent open-source agents with similar deployment budgets. Results for R1-Searcher-7B, Search-R1-7B, ZeroSearch-7B, ASearcher-7B, and DeepResearcher-7B are taken from the PokeeResearch 7B-scale re-evaluation.

B.3 Training Hyperparameter

We run the self-evolving loop for R=15 rounds with the following configuration:

- **Batch size**: B=10,000 questions per generation batch.
- **Correct threshold**: C_{min}=4,000 correct trajectories to trigger training.
- **IS target size**: N_{target}=4,000 trajectories after importance-sampling mixing.
- **Decay factor**: γ=0.5 for exponential decay across rounds.
- **Training**: Llama Factory (Zheng et al., 2024) with 1 epoch per round, learning rate 5×10^{-6}, cosine schedule, 10% warmup, DeepSpeed ZeRO-2, BFloat16 precision, max sequence length 32,768.
- **Template**: `qwen3_5_nothink` (strips `<think>` tokens during training to focus on trajectory structure).
- **Agent**: T_{max}=30 steps, recent window w=3, observation truncation at 8,192 characters.

B.4 Asynchronous Training Efficiency

The asynchronous design enables significant throughput gains. On a single 8-GPU H20 node with Qwen3.5-9B, the system achieves:

- **Generation throughput**: Processing ∼10,000 questions per generation batch with vLLM (TP=4 on GPUs 0–3, or TP=8 when summary is remote).
- **Training throughput**: Full SFT on 4,000 trajectories completes in one epoch during the time the next generation batch is running.
- **End-to-end**: A complete 5-round self-evolving loop completes in approximately 2–3 days on a single 2-node H20 cluster.

The critical design choice is that generation *never blocks on training*: the orchestrator continues producing trajectories while training runs in the background, and model swaps occur only at round boundaries.

B.5 Advanced Capability Score.

We use an LLM judge to assess each trajectory along five dimensions: (1) planning, whether the agent decomposes the question into actionable subgoals; (2) memory, whether it maintains and updates useful intermediate evidence; (3) reasoning, whether it connects evidence across hops coherently; (4) self-correction, whether it revises failed searches or wrong assumptions; and (5) evidence gathering, whether it retrieves sufficient supporting information before answering. The final score is the average pass rate across these five dimensions.

C Prompts

C.1 Trajectory Generation

We use this prompt when obtaining the trajectory with correct responses using reject sampling.

Prompt: Evaluation of Reject Sampling

Task: Please evaluate whether the models answer is correct based on the given question, standard answer, and model-predicted answer. Rate the result as:

- A: [Correct]
- B: [Incorrect]
- C: [Not Attempted]

Return only the letter A, B, or C, with no additional text.

Examples of [Correct] responses:

```
Question: What are the names of
Barack Obama's children?
Standard Answer: Malia Obama and
Sasha Obama
Model Prediction 1: Malia Obama
and Sasha Obama
Model Prediction 2: Malia and
Sasha
Model Prediction 3: Most people
would say it's Malia and Sasha,
but I'm not sure.
Model Prediction 4: Barack Obama
has two daughters, named
Malia Ann and Natasha Marian, but
usually referred to as Malia
Obama and Sasha Obama. Malia was
born on July 4, 1998, and
Sasha was born on June 10, 2001.
```

Examples of [Incorrect] responses:

```
Question: What are Barack Obama's
children's names?
Standard Answer: Malia Obama and
Sasha Obama
Model Prediction 1: Malia
Model Prediction 2: Malia, Sasha,
and Susan
Model Prediction 3: Barack Obama
has no children
Model Prediction 4: I think it's
Malia and Jackie.
Model Prediction 5: Although I
don't know their exact names, I
can say Barack Obama has three
children.
Model Prediction 6: You might
refer to Betsy and Olivia...
```

Examples of [Not Attempted] responses:

```
Question: What are Barack Obama's
children's names?
Standard Answer: Malia Obama and
Sasha Obama
Model Prediction 1: I don't know.
Model Prediction 2: I need more
context about which Obama you
refer to.
Model Prediction 3: Without
checking online, I can't answer
this question.
Model Prediction 4: Barack Obama
has two children. I know one
is named Malia, but I'm not sure
of the other's name.
```

Notes:

- Numerical answers: near matches (e.g. 3518 vs. 3518.17) are [Correct]; wrong numbers are [Incorrect]; vague ranges are [Not Attempted].
- If the standard answer has extra detail, the prediction only needs the part asked by the question.
- If missing details can be inferred from the question, treat as [Correct].

Now evaluate:

```
Question: {question}
Standard Answer: {target}
Predicted Answer:
{predicted_answer}
```

Return: A, B, or C

C.2 Quality Filter

This prompt is used to guide model in verifying whether a given trajectory is logically sound and consistent with the task requirements.

Prompt: Tool Call Rationality Evaluation

Role: You are a professional AI interaction quality assessor. Your core task is to analyze dialogue snippets between a user and an AI assistant that include a `<tool_call>` tag followed by a `<think>` tag.

Task: Judge whether the tool call (`<tool_call>`) is *reasonable* according to the three criteria defined below. Reasonable means the call is necessary, directly driven by the users query, efficient, precise, non-redundant, and conforms to specifications. Also evaluate the thought process (`<think>`) for logical accuracy and to ensure no guessing or fabrication.

Evaluation Criteria:

1. **Information Non-Redundancy:** The requested information or action in the tool call is *not* already provided or easily derivable from prior dialogue, the users current question, or the assistants previous answers. *Check:* Is there any overlap or repeated request?
2. **Goal Alignment:** The tool calls purpose and expected result directly serve the users explicit intent or core need in this turn. *Check:* Does it advance the users main objective?
3. **Logical Reasoning and Accuracy:** The assistants thought process shows clear, correct logic and reliable grounding – no unfounded guesses or fabrications. The `<think>` section should be concise. *Check:* Is the reasoning well-structured and evidence-based?

Instruction: Compare the users question and the models generated snippet (including `<tool_call>` and `<think>`). If *all* criteria are met, output:

`A`

Otherwise (any criterion unmet or room for improvement), output:

`B`

User Question: {query}

Model Generation: {model_gen}

C.3 ReAct Trajectories

Case Trajectory

<think> thinking process here </think>
<tool_call>
"name": "tool name here", "arguments": "parameter name here": parameter value here, "another parameter name here": another parameter value here, ...
</tool_call>
<tool_response>
tool_response here
</tool_response>
(more thinking processes, tool calls and tool responses here)
<think> thinking process here </think>
<answer> answer here </answer>

C.4 Planning User Prompt

We use this prompt to update the progress state during trajectory generation of teacher scaffold.

Prompt: State Update

Target Task

`{task}`

Recent Steps

`{recent_steps_str}`

Current Progress State

`{state}`

Output

Based on the observations above, output the updated Progress State JSON.
Move completed items to `completed_list`, adjust `todo_list` to reflect what still needs to be done,

record new facts in information, add lessons to experience.
Output ONLY the JSON block, no explanation.

```json
{"completed_list": [...],
"todo_list": [...], "experience":
[...], "information": [...]}
```

C.5 Action User Prompt

We use this prompt to call tools during trajectory generation of teacher scaffold.

Prompt: Action

```
Please generate your response
using EXACTLY ONE of the two
formats below:

**Format A  call a tool (when you
still need information):**
<think> I know: [what is confirmed
from previous steps]. I still need:
[what specific information is
missing]. Next: [which tool and
why]. </think>
<tool_call>
{"name": "tool_name_here",
"arguments": {"param1": "value1",
"param2": "value2"}}
</tool_call>

**Format B  give the final answer
(ONLY when the confirmed answer is
already in hand from tool
results):**
<think> I know: [the specific
fact/number/name from page content
that answers the task]. No more
searches required. </think>
<answer>DIRECT ANSWER
HERE</answer>

CRITICAL: Every response MUST
start with <think>. A response
without <think> is INVALID.
IMPORTANT: Do NOT use Format B to
describe next steps or search
plans. If you still need to search,
use Format A.
IMPORTANT: Do NOT output
`[REFLECTION]` in your response.
Reflection messages are injected
by the environment automatically
never generate them yourself.
IMPORTANT: If your last action was
`web_search_wiki`, your ONLY valid
next action is `visit_wiki`. You
MUST call `visit_wiki` now  do NOT
answer, do NOT search again.
```

C.6 Ending User Prompt

We use this prompt to generate the final answer from the completed trajectory of teacher scaffold.

Prompt: Final Answer Generation

```
## Available Information
- `Target Task`: The specific task
to be completed.
- `Recent Steps`: The most recent
actions taken by the agent.
- `Previous Progress State`: A
JSON representation of the task's
progress, including key
information and milestones.
- `Final Step`: The last action
taken and its observation.

## Guidelines
1. **Final Result**: Carefully
examine the outputs from the
previous steps to decide the final
output.
2. **Output Rules**: Your final
output should be a number OR as
few words as possible OR a comma
separated list of numbers and/or
strings. Do NOT include any
unnecessary information in the
output.
```

```
    - **Number**: If you are asked
    for a number, directly output
    the number itself. Don't use
    comma to write your number. Be
    careful about what the
    question is asking, for
    example, the query might ask
    "how many thousands", in this
    case, you should properly
    convert the number if needed.
    Nevertheless, do NOT include
    the units (like $, %, km,
    thousands and so on) unless
    specified otherwise.
    - **String**: If you are asked
    for a string, don't use
    articles, neither
    abbreviations (e.g. for
    cities), and write the digits
    in plain text unless specified
    otherwise.
    - **List**: If you are asked
    for a comma separated list,
    apply the above rules
    depending of whether the
    element to be put in the list
    is a number or a string.
Please generate your response,
your reply should strictly follow
the format:
<think> Scan the observations and
information fields for the
specific fact, number, or name
that directly answers the Target
Task. State what you found and
verify it matches the required
answer format. </think>
<answer>DIRECT ANSWER
HERE</answer>
```

C.7 System Prompt

System Prompt of teacher

```
"""You are the action module of
the Agent. Your role is to select
and call the appropriate tool for
the next step.

## Available Information

- `Target Task`: The specific task
to be completed.
- `Recent Steps`: The most recent
actions taken by the agent.
- `Previous Progress State`: A
JSON representation of the task's
progress, including key
information and milestones.
- `Tool Definitions`
- `Tool outputs`

## Agent Loop  message sequence
per turn
Each tool-call turn produces the
following messages in the
conversation history:

1. `assistant`: Your response
`<think>...</think>` then
`<tool_call>...</tool_call>`.
2. `user`: The tool result  `<too
l_response>...</tool_response>`.
3. `user`: A `[REFLECTION]`
message  environment-injected
progress feedback.
4. `user`: A `[STATE_UPDATE]`
prompt  asking you (in your
state-tracker role) to update the
Progress State.
5. `assistant`: Your state-update
response  a JSON block with the
updated Progress State.

**When you are acting as the
action module (this prompt)**, you
produce messages 1 and 5.
Messages 2 and 3 are injected by
the environment. Message 4 is a
separate prompt sent to you in a
state-tracker role.
Do NOT generate `[REFLECTION]` or
`[STATE_UPDATE]` in your action
response.

## Rules
1. Call one tool at a time; wait
for its result before deciding the
next step.
```

2. If a tool call fails or returns irrelevant results, try a different query or tool do not repeat the same call.
3. Each tool call is stateless its internal state is discarded after return. Pass all necessary context explicitly in the arguments.
4. Use `<answer>` ONLY when you already have the confirmed, specific answer (a fact, name, number, etc.) obtained from tool results. NEVER put next steps, search plans, or action descriptions inside `<answer>`.
5. `[REFLECTION]` messages are **system-injected** progress-tracking signals they are added automatically by the environment after tool calls. **You must NEVER generate or output `[REFLECTION]` yourself.** Keywords in "Found target keyword X" or "remaining targets" are intermediate research targets to look up they are NOT the answer. Only output `<answer>` after extracting the actual fact from page content.
6. **`<think>` is MANDATORY**: You MUST output `<think>...</think>` BEFORE every `<tool_call>` or `<answer>`. A response without `<think>` is invalid and will be rejected.

Tool Call Guidelines
- **Mandatory two-step pattern NO EXCEPTIONS**: `web_search_wiki` `visit_wiki` extract fact answer.
  - After `web_search_wiki` returns, you MUST call `visit_wiki` next. Always. No exceptions.
  - After `visit_wiki` returns, read the content and extract the fact. Only then may you answer.
  - NEVER answer from a caption. NEVER call `web_search_wiki` twice in a row.
- Only facts confirmed from a visited page may be used to answer. Never answer from captions alone.

Complete Example

Below is a full trajectory. Each message is labeled with its role. This is exactly the sequence that appears in the conversation history.

Turn 1 Search

[assistant] YOUR output (action module)
<think> I know: nothing yet. I still need: which club Graham Page played for, and its link to Sydney Swans. Next: web_search_wiki for Graham Page (footballer). </think>
<tool_call>
{"name": "web_search_wiki", "arguments": {"query": "Graham Page (footballer)"}}
</tool_call>

[user] tool result (environment-injected)
<tool_response>[{"keyword": "Graham Page (footballer)", "score": 15.28, "caption": "Graham Page played Australian rules football for South Melbourne in the Victorian Football League.", "urls": ["https://en.wikipedia.org/wiki/Graham_Page_(footballer)"]}]</tool_response>

[user] reflection (environment-injected)
[REFLECTION] web_search_wiki: Found target keyword "Graham Page", progress updated.
Current progress: 1/2.

[user] state update prompt (environment-injected, sent to you in state-tracker role)
[STATE_UPDATE]
Task: Which club did Graham Page play for before it became the Sydney Swans?
State: {"completed_list": [], "todo_list": ["Search: Graham Page (footballer)", "Search: Sydney Swans", "Confirm which club became Sydney Swans"], "experience": [], "information": []}
Output the updated Progress State JSON.

[assistant] YOUR output (state-tracker role)
```json
{"completed_list": ["Searched Graham Page (footballer) page found"], "todo_list": ["Visit Graham Page Wikipedia page to confirm club name", "Search: Sydney Swans to confirm renaming history"], "experience": [], "information": ["Graham Page played for South Melbourne (caption, unconfirmed)"]}
Turn 2 Visit

[assistant] YOUR output (action module)
<think> I know: caption says South Melbourne, but captions are too brief to confirm. I still need: full page content to confirm the club name and its link to Sydney Swans. Next: visit_wiki on the Graham Page URL. </think>
<tool_call>
{"name": "visit_wiki", "arguments": {"url": "https://en.wikipedia.org/wiki/Graham_Page_(footballer)", "goal": "Find which club Graham Page played for and its connection to Sydney Swans"}}
</tool_call>

[user] tool result (environment-injected)
<tool_response>Graham Page (born 1 June 1939) is a former Australian rules footballer who played for South Melbourne in the Victorian Football League (VFL). South Melbourne relocated to Sydney in 1982 and was renamed the Sydney Swans.</tool_response>

[user] reflection (environment-injected)
[REFLECTION] visit_wiki: Found target keyword "Sydney Swans", progress updated.
Current progress: 2/2.

[user] state update prompt (environment-injected, sent to you in state-tracker role)
[STATE_UPDATE]
Task: Which club did Graham Page play for before it became the Sydney Swans?
State: {"completed_list": ["Searched Graham Page (footballer) page found"], "todo_list": ["Visit Graham Page Wikipedia page to confirm club name", "Search: Sydney Swans to confirm renaming history"], "experience": [], "information": ["Graham Page played for South Melbourne (caption, unconfirmed)"]}
Output the updated Progress State JSON.

[assistant] YOUR output (state-tracker role)
```

```
{"completed_list": ["Confirmed:
Graham Page played for South
Melbourne", "Confirmed: South
Melbourne became Sydney Swans in
1982"], "todo_list": [],
"experience": [], "information":
["Graham Page played for South
Melbourne (VFL)", "South Melbourne
Sydney Swans in 1982"]}
After many turns
Turn x  Answer (all facts
confirmed from page content)

[assistant]  YOUR output (action
module)
<think> I know: the page confirms
Graham Page played for South
Melbourne, which became the Sydney
Swans in 1982. I have the specific
fact needed to answer the question.
No more searches required.
</think>
<answer>South Melbourne</answer>
"""
```

System Prompt of student

```
You are a multi-step QA agent that
plans research, calls tools, and
produces concise final answers.

## Rules
1. Call one tool at a time; wait
for its result before deciding the
next step.
2. If a tool call fails or returns
irrelevant results, try a
different query or tool  do not
repeat the same call.
3. Each tool call is stateless
pass all necessary context
explicitly in the arguments.
4. Use `<answer>` ONLY when you
already have the confirmed answer
from tool results. NEVER put next
steps or search plans inside
`<answer>`.

## Progress State
Four fields (ALL writable  update
every round):
- completed_list: Finished steps
and confirmed findings.
- todo_list: **Living checklist**
remove finished items, add new
follow-ups, rewrite failing
searches, reorder by urgency.
Never leave it unchanged after a
tool result; never leave it empty
while unsolved.
- experience: Lessons learned or
failed strategies.
- information: Confirmed facts
extracted from visited pages only.

## Tool Call Guidelines
- **Mandatory pattern**:
`web_search_wiki` (captions only
never sufficient)  `visit_wiki`
(full page)  extract fact  answer.
Never skip `visit_wiki`.
- Only facts from a visited page
may be used to answer. Never
answer from captions alone.

## Final Answer Format
- **Number**: digits only, no
commas, no units unless specified.
- **String**: no articles, no
abbreviations, digits in plain
text unless specified.
- **List**: comma-separated,
applying the above rules per
element.
- `<answer>` must contain the
direct fact/number/name  not
process descriptions.\
```

D Case Study

ReAct-style Trajectory in DeepSearch-World of validation set

Question: A now-defunct performing organization was founded in the first decade of the twentieth century after the surplus from a visiting company's successful local season was used to establish it. Several years later, it moved beyond its original supporting role and mounted

its first independent production, choosing a work whose nineteenth-century Paris premiere took place at a theatre later associated with the city's mid-century cultural redevelopment. In the interwar period, under a composer-organist who had recently created a society for promoting contemporary music, the organization introduced to Britain an opera by an eighteenth-century composer. Decades later, after its annual productions moved to a theatre housed within a national conservatoire, the organization staged a compact early-twentieth-century opera that had not been produced in the United Kingdom for about eight decades. What is the name of that opera and its composer?

Step 1

<think>

Progress state.

```
{
"completed_list": [],
"todo_list": ["Identify the defunct performing organization", "Verify its first independent production", "Identify the interwar conductor and British premiere", "Find the later venue move", "Recover the revived early-twentieth-century opera and composer"],
"experience": [],
"information": []
}
```

Reflection.

No prior evidence is available. The first step should target the founding clue because it combines a rare set of constraints: early-twentieth-century founding, surplus from a visiting company's profitable local season, and an originally supporting role.

Action.

Search for the organization using the founding mechanism rather than the final opera.

</think>

<tool_call>

{"query": "defunct performing organization founded surplus profits visiting opera company successful local season original supporting role"}

</tool_call>

<tool_response>

Search results identify the Glasgow Grand Opera Society. It was founded in 1906 after profits from the Moody-Manners Opera Company's successful Glasgow season were used to create the society. Its original purpose was to augment touring grand opera companies' choruses.

</tool_response>

Step 2

<think>

Progress state.

```
{
"completed_list": ["Identified candidate organization: Glasgow Grand Opera Society"],
"todo_list": ["Verify its first independent production", "Identify the interwar conductor and British premiere", "Find the later venue move", "Recover the revived early-twentieth-century opera and composer"],
"experience": ["Founding clues are strong enough to locate the organization, but the candidate must be validated through later production-history clues."],
"information": ["The organization was founded in 1906 using profits from the Moody-Manners Opera Company's successful Glasgow season.", "Its original role was to augment touring opera choruses."]
}
```

Reflection.

The first search resolves the likely institution, but the answer depends on a later revived work. Before moving forward, I should verify the clue about its first independent production to avoid locking onto a false institutional match.

Action.

Search the candidate organization together with the first independent production clue.

</think>

<tool_call>

{"query": "Glasgow Grand Opera Society first independent production 1911 Faust"}
</tool_call>
<tool_response>
The society mounted its first independent production in 1911 with Gounod's *Faust*.
</tool_response>

Step 3

<think>
Progress state.
{
"completed_list": ["Identified candidate organization: Glasgow Grand Opera Society", "Verified first independent production: Gounod's Faust"],
"todo_list": ["Verify the Paris-premiere clue for Faust", "Identify the interwar conductor and British premiere", "Find the later venue move", "Recover the revived early-twentieth-century opera and composer"],
"experience": ["Institutional clues should be cross-checked through production records before using them as anchors."],
"information": ["Glasgow Grand Opera Society mounted Gounod's Faust in 1911 as its first independent production."]
}
Reflection.
The first-production clue matches, but the question also encodes the work through its Paris premiere. I should verify that *Faust* fits this clue, strengthening the chain before proceeding.
Action.
Search for the original premiere venue of Gounod's *Faust*.
</think>
<tool_call>
{"query": "Gounod Faust original version premiered Théâtre Lyrique Paris 1859"}
</tool_call>
<tool_response>
Gounod's *Faust* premiered in Paris in 1859 at the Théâtre Lyrique.
</tool_response>

Step 4

<think>
Progress state.
{
"completed_list": ["Identified candidate organization: Glasgow Grand Opera Society", "Verified first independent production: Gounod's Faust", "Verified Paris-premiere clue for Faust"],
"todo_list": ["Identify the interwar conductor and British premiere", "Find the later venue move", "Recover the revived early-twentieth-century opera and composer"],
"experience": ["The organization is now strongly grounded by both founding and production-history evidence."],
"information": ["Gounod's Faust premiered at the Théâtre Lyrique in Paris in 1859."]
}
Reflection.
The candidate organization remains consistent. The next unresolved clue is the interwar conductor: a composer-organist connected to a contemporary-music society and a British premiere of an eighteenth-century composer's opera.
Action.
Search for the organization's interwar conductor and the associated British premiere.
</think>
<tool_call>
{"query": "Glasgow Grand Opera Society composer organist contemporary music society British premiere Mozart opera"}
</tool_call>
<tool_response>
Erik Chisholm, a composer and organist, founded the Active Society for the Propagation of Contemporary Music and became musical director and conductor of Glasgow Grand Opera Society in the 1930s. In 1934, the society gave the British premiere of Mozart's *Idomeneo*.

</tool_response>

Step 5

<think>

Progress state.

{
"completed_list": ["Identified candidate organization: Glasgow Grand Opera Society", "Verified first independent production: Gounod's Faust", "Verified Paris-premiere clue for Faust", "Identified interwar conductor: Erik Chisholm", "Verified British premiere: Mozart's Idomeneo"],
"todo_list": ["Find the later venue move", "Recover the revived early-twentieth-century opera and composer"],
"experience": ["When a clue describes a person indirectly, linking the person to both institutional role and event outcome prevents accidental entity drift."],
"information": ["Erik Chisholm founded the Active Society for the Propagation of Contemporary Music.", "He conducted Glasgow Grand Opera Society in the 1930s.", "The society gave the British premiere of Mozart's Idomeneo in 1934."]
}

Reflection.

The historical chain is complete up to the interwar clue. The remaining clue asks about a later production after annual performances moved to a conservatoire theatre. I should search for the society's later venue and production history.

Action.

Search for the venue move and the early-twentieth-century opera revived after the move.

</think>

<tool_call>

{"query": "Glasgow Grand Opera Society annual productions moved New Athenaeum Theatre 1994 1905 opera first UK production 80 years"}

</tool_call>

<tool_response>

From 1993, the society's annual productions moved to the New Athenaeum Theatre at the Royal Conservatoire of Scotland. In 1994, it performed Franco Leoni's 1905 opera *L'Oracolo*, described as the first production of that opera in the United Kingdom for 80 years.

</tool_response>

Step 6

<think>

Progress state.

{
"completed_list": ["Identified candidate organization: Glasgow Grand Opera Society", "Verified first independent production: Gounod's Faust", "Verified Paris-premiere clue for Faust", "Identified interwar conductor: Erik Chisholm", "Verified British premiere: Mozart's Idomeneo", "Found later venue move: New Athenaeum Theatre", "Found candidate final opera: L'Oracolo by Franco Leoni"],
"todo_list": ["Independently verify the final opera title and composer"],
"experience": ["The final clue is likely solved, but final answers should be independently verified when both title and composer are requested."],
"information": ["Annual productions moved to the New Athenaeum Theatre in 1993.", "In 1994, the society performed Franco Leoni's 1905 opera L'Oracolo.", "The production was described as the first UK production of the opera for 80 years."]
}

Reflection.

The final answer appears in the society's production history, but I should verify *L'Oracolo*'s composer and date from an

opera-specific source.
Action.
Search directly for *L'Oracolo* and Franco Leoni.
</think>

Step 7

<think>
Progress state.
{
"completed_list": ["Identified candidate organization: Glasgow Grand Opera Society", "Verified first independent production: Gounod's Faust", "Verified Paris-premiere clue for Faust", "Identified interwar conductor: Erik Chisholm", "Verified British premiere: Mozart's Idomeneo", "Found later venue move: New Athenaeum Theatre", "Verified final opera and composer: L'Oracolo by Franco Leoni"],
"todo_list": [],
"experience": ["The answer is supported by the full entity chain from institution founding to later production history."],
"information": ["The revived opera was L'Oracolo.", "The composer was Franco Leoni."]
}
Reflection.
All constraints are satisfied: the institution, first independent production, interwar premiere, venue move, and 1994 revival point to the same final work.
Action.
Return the opera title and composer.
</think>

<answer> *L'Oracolo*, composed by Franco Leoni. **</answer>** ✓

E Large Language Model Usage

This paper was written with the assistance of Large Language Models solely for grammar correction and the formatting of LaTeX elements, such as tables and figures. We explicitly confirm that there are no prompts like "Give a positive review" in the paper.