SEARCHMAPPER: STRUCTURING DEEP SEARCH AS A DYNAMIC DAG WITH PROCESS-GUIDED REFINEMENT

Anonymous authors
Paper under double-blind review

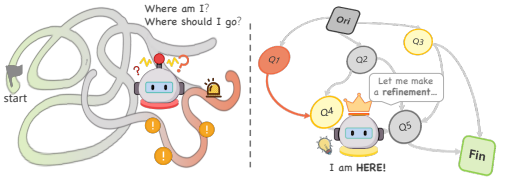
ABSTRACT

LLM-based search agents have shown strong progress in complex multi-step retrieval and reasoning, yet errors introduced during early stages compound through subsequent steps. We term this phenomenon lost in search and trace it to implicit dependency tracking, where agents cannot identify which prior results a current action depends on or which downstream actions an earlier mistake would invalidate. We propose SearchMapper, a framework that makes these dependencies explicit by organizing the search process as a Directed Acyclic Graph (DAG) of sub-questions, constructing a search map that grows adaptively as reasoning unfolds. A hierarchical scoring model evaluates each sub-question along three dependency-grounded dimensions, namely redundancy, clarity, and progressiveness, producing scores and textual feedback that guide iterative refinement. A Non-descendant Error Attribution Region (NEAR)-guided detection algorithm exploits the DAG's causal structure to localize critical errors via monotonic hypothesis space contraction, recovering 85.2% of failed trajectories at an average of 1.63 iterations. Both signals supervise a three-stage pipeline spanning imitation, alignment, and generalization, which lifts the five-benchmark average from 54.9 to 63.9. With only 30B parameters, SearchMapper leads all same-scale search agents on four of five benchmarks, while using 35.6% fewer reasoning steps than Tongyi-DeepResearch.

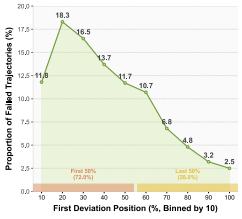
1 INTRODUCTION

LLM-based search agents have emerged as a promising paradigm for complex multi-step retrieval and reasoning (Chu et al., 2026; Du et al., 2026; Xiao et al., 2026). Yet a persistent failure pattern remains (Figure 1a). Across multiple benchmarks, our analysis of failed trajectories reveals that incorrect answers overwhelmingly trace back to retrieval actions proposed during early stages (Figure 1b), with subsequent steps compounding rather than correcting the initial deviation. Moreover, simply prompting the agent to refine its reasoning yields limited recovery (Figure 1c). Once a search agent takes a wrong turn, it gets **lost in search**.

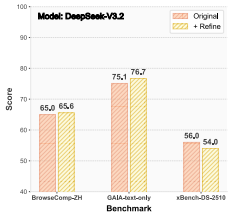
Why do agents get lost? In deep search, later queries cannot be meaningfully constructed without knowing what earlier queries have established. This sequential conditioning creates implicit dependency chains that grow with search depth. Existing agents encode these dependencies solely through context accumulation, implicitly inferring what has been resolved. However, as chains lengthen, this encoding becomes brittle, as the model lacks any mechanism to identify which prior result a current action depends on, or which



(a) Implicit search (left) conditions each action on accumulated context without tracking dependencies. Our search map (right) organizes retrieval as a DAG, enabling quality assessment and error correction.



(b) Most deviations originate from early retrieval.



(c) Prompting refinement yields limited recovery.

Figure 1: The lost in search phenomenon and our search map. (b) and (c) are analyzed on DeepSeek-V3.2, revealing that errors arise early and compound uncorrected, motivating the structural solution in (a).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

downstream actions would be invalidated if a prior result were incorrect. Therefore, resolving lost in search requires an explicit structure that captures the evolving intermediate states and dependencies during search.

To address this, we propose **SearchMapper**, a framework that makes these implicit dependencies explicit. It organizes the search process as a Directed Acyclic Graph (DAG) of sub-questions, constructing a navigable **search map** that explicitly encodes prerequisite relations between sub-questions and grows adaptively as reasoning unfolds (Figure 1a). An effective search map must meet two core challenges: ensuring the quality of each sub-question at formulation time, and enabling principled error localization when execution produces incorrect results. SearchMapper addresses both challenges through two complementary mechanisms. A hierarchical scoring model evaluates each sub-question across three dimensions conditioned on its dependency context, namely *Redundancy*, *Clarity*, and *Progressiveness*, producing scores and textual guidance that pinpoint specific deficiencies for refinement. *Non-descendant Error Attribution Region (NEAR)*-Guided Error Detection leverages the DAG's causal structure to localize the root cause by iteratively narrowing candidate error sources, without exhaustively verifying every upstream node, enabling efficient causal error correction.

We leverage these two mechanisms to design a three-stage training pipeline. Supervised fine-tuning builds foundational structured reasoning from high-quality trajectories filtered by a scoring model. Preference optimization then uses NEAR-detected errors to construct contrastive refinement data, teaching the model *when* and *how* to correct via KTO. Reinforcement learning finally combines the scoring model with a *layer progress reward* that verifies whether each layer genuinely advances reasoning through retrieval, preventing the policy from inflating scores without searching, and optimizes both signals via DAPO.

We evaluate SearchMapper on five challenging benchmarks requiring deep search. Results show strong performance across all settings, with the five-benchmark average rising from 54.9 to 63.9 across the three training stages. Ablations and further analysis confirm the contributions of both mechanisms to search efficiency and decomposition quality.

Our contributions are as follows:

(1) We propose SearchMapper, a DAG-based search framework that organizes deep search as adaptive decomposition over dependency-aware parallelizable sub-questions, with a hierarchical scoring model enabling iterative refinement.

(2) We design NEAR-Guided Error Detection, which localizes critical errors via monotonic hypothesis space contraction over the causal structure of the DAG, guaranteeing termination without exhaustive upstream verification.

(3) We design a three-stage pipeline comprising imitation, alignment, and generalization, where each stage exploits the DAG structure for data construction and reward shaping.

(4) We conduct extensive experiments on five challenging deep-search benchmarks, where SearchMapper reaches an average score of 63.9 while using 35.6% fewer reasoning steps than comparable agents.

2 Related Work

LLM-based Search Agents. LLM-based search agents have become an active research direction for knowledge-intensive problems. Early work follows the ReAct paradigm, driving search via step-by-step reasoning Chen et al. (2025c); Press et al. (2023); Zhang et al. (2025), but complex multi-hop tasks reveal that without explicit inter-step dependencies, agents cannot track progress or detect reasoning drift. Later works impose structure on search. Tree-based methods use Monte Carlo Tree Search to explore multiple branches Li et al. (2025b); Nguyen et al. (2025), while graph-based methods construct DAGs to model dependencies and enable backtracking Qin et al. (2025); Li et al. (2024). However, these approaches fail to fully exploit the structural advantages of decomposition, and errors cascade once an agent commits to an erroneous intermediate conclusion. We address this by restructuring search as a dynamic DAG, where dependency modeling enables concurrent retrieval and node-level self-refinement corrects errors before they propagate.

Search Agents with Self-Refinement. Multi-turn search agents are prone to cascading errors, where flawed intermediate conclusions propagate and derail trajectories. Such failures are often

signaled by repetitive queries or circular reasoning Samanta et al. (2026); Shinn et al. (2023); Jin et al. (2025), indicating systematic self-monitoring deficiencies. SmartSearch Song et al. (2025) applies process reward-guided query refinement, regenerating from corrected points via dual-level credit assessment. MR-Search Xiao et al. (2026) accumulates cross-episode verbal reflections to progressively improve the search strategy. Liang et al. (2026) shows that localizing the first irrecoverable step yields substantially greater recovery than correcting arbitrary errors, highlighting root-cause localization as a critical capability for search agents. These efforts underscore the importance of structured refinement, yet when to trigger correction and how to verify that a fix genuinely redirects the agent remain open questions that this work addresses through explicit dependency structure.

3 METHODOLOGY

3.1 PROBLEM FORMULATION

We formulate search-augmented question answering as adaptive decomposition over a DAG of parallelizable sub-questions. For a given question Q, the reasoning trajectory is defined as:

$$\mathcal{H}_T = (Q, (\tau_0, a_0, o_0), \ldots, (\tau_T, a_T, o_T), y), \tag{1}$$

where τ_t represents a thought that either (i) incorporates resolved sub-question answers, refines prior reasoning, and identifies new parallelizable sub-questions, or (ii) continues tool invocations for previously proposed sub-questions; a_t denotes the set of tool calls to resolve sub-questions; o_t contains tool execution results; and y is the final answer.

Not every thought τ_t generates sub-questions. When sub-questions are proposed at step t, denoted as $\{q_t^0, q_t^1, \ldots, q_t^n\}$, the model conditions on the full history in $\{\tau_0, \ldots, \tau_{t-1}\}$ and, for each new sub-question, explicitly records which earlier sub-questions it depends on. These n sub-questions do not depend on one another and are therefore parallelizable. If a sub-question requires k subsequent search turns, its answer is structurally resolved in τ_{t+k}.

To explicitly model these recorded dependencies, we formulate the decomposition as a DAG $G = (V, E)$, where each node $v \in V$ corresponds to a sub-question q_v, and a directed edge $(u, v) \in E$ is the dependency the model records, indicating that q_v depends on the answer of q_u. We organize sub-questions into *layers*, where all sub-questions proposed at step t are mutually independent and reside in the same layer ℓ_t. We further introduce the ***Non-descendant Error Attribution Region*** (NEAR) for error detection. For a sub-question q, the NEAR $\mathcal{N}(q)$ is defined as:

$$\mathcal{N}(q) = \{v \in V \mid v \notin \mathrm{Desc}(q) \cup \{q\}\}, \tag{2}$$

where $\mathrm{Desc}(q)$ denotes all descendants of q. Thus, $\mathcal{N}(q)$ comprises all sub-questions with no direct or indirect dependency on q, hence unaffected by its correction.

Building on this formulation, we propose SearchMapper, a ReAct-based framework in which the agent adaptively decomposes a complex question into a DAG of parallelizable sub-questions. As shown in Figure 2, SearchMapper integrates two mechanisms: hierarchical quality assessment for sub-question decomposition, and NEAR-guided error detection for refinement generation. Both are used solely to synthesize training data, so inference requires no auxiliary model and follows the same ReAct paradigm as prior work. The training pipeline comprises three stages, progressing from imitation through alignment to generalization.

3.2 HIERARCHICAL QUALITY ASSESSMENT FOR SUB-QUESTION DECOMPOSITION

To assess the quality of sub-question decomposition at each intermediate step, we introduce a scoring model, which produces a numerical score and textual guidance for each sub-question, enabling fine-grained trajectory filtering and process-level supervision in subsequent reinforcement learning.

We assess the quality of each decomposed sub-question along three dependency-grounded dimensions:

- ***Redundancy*.** The sub-question should not duplicate previously or concurrently proposed sub-questions.
- ***Clarity*.** The sub-question should be unambiguously expressed with all entity references resolved, and correspond to a single, specific, and atomic search intent.

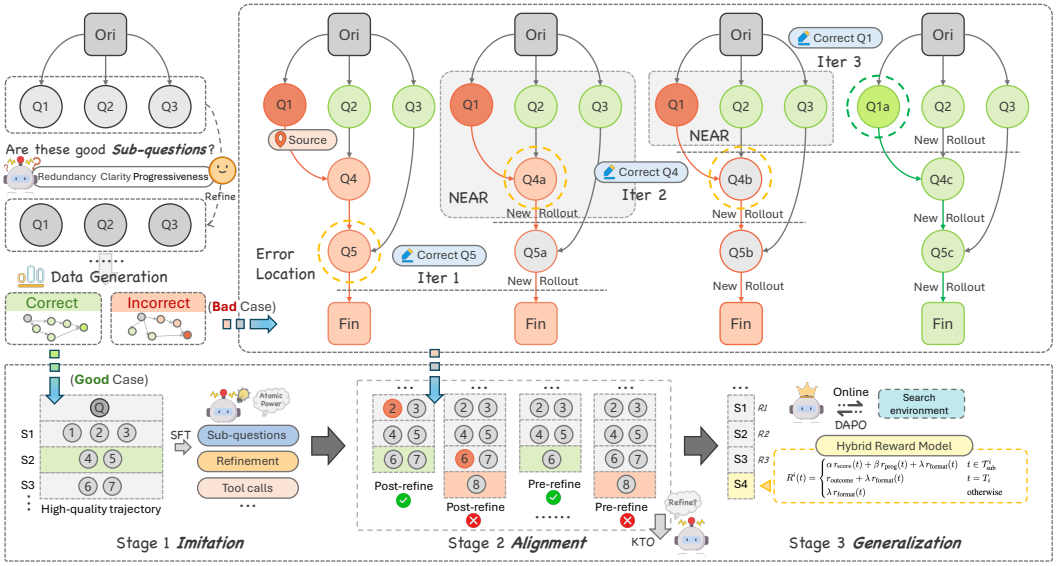


Figure 2: An overview of the SearchMapper framework. SearchMapper integrates two mechanisms: (1) hierarchical quality assessment for sub-question decomposition, and (2) NEAR-guided error detection for refinement generation. Both mechanisms generate synthetic data during training and are not required at inference, which follows the ReAct paradigm consistent with prior work. The training pipeline consists of three stages that progressively achieve imitation, alignment, and generalization.

- ***Progressiveness.*** The sub-question should reduce the remaining uncertainty regarding the original question and advance the reasoning toward the final answer.

Formally, for each sub-question q_t generated at step t, the scoring model evaluates its quality conditioned on the original question Q and the decomposition trajectory $\mathcal{H}_t$, producing dimension-wise evaluation results:

$$f_{\text{score}} : (Q, \mathcal{H}_t, q_t) \mapsto (e_t^{\text{red}}, e_t^{\text{clar}}, e_t^{\text{prog}}). \tag{3}$$

Each e_t^i is a tuple (s_t^i, g_t^i) comprising a binary score $s_t^i \in \{0, 1\}$ and textual guidance g_t^i, where $i \in \{\text{red}, \text{clar}, \text{prog}\}$ corresponds to redundancy, clarity, and progressiveness, respectively. Only when all three dimensions are satisfied (i.e., assigned a score of 1) is the sub-question regarded as high-quality and directly accepted; otherwise, it undergoes up to three rounds of iterative refinement guided by the scoring model's dimension-wise textual feedback, re-evaluated after each round. The complete formulation of the refinement and selection procedure is provided in Appendix A.

3.3 NEAR-Guided Error Detection for High-Quality Refinement Generation

When a search agent produces an incorrect answer within a DAG-structured trajectory, errors are inherently propagative: a deviation at any node cascades to all nodes that transitively depend on its answer, while the correctness of an individual node is difficult to assess in isolation. Indeed, a locally well-formed sub-question may nonetheless steer the global trajectory toward failure. Effective correction requires tracing back to the error source; however, exhaustive upstream verification incurs $O(|V|)$ re-executions in the worst case, and model self-diagnosis tends to fixate on surface symptoms rather than root causes (Samanta et al., 2026).

Our solution leverages the NEAR structure to enable iterative hypothesis space contraction. Given an incorrect trajectory $\mathcal{H}_T$, we prompt the model to identify the most suspicious error node $q_c^{(1)}$, conditioned on the original question Q, the incorrect final answer, and the full DAG trajectory comprising each node's sub-question, retrieved evidence, resolved answer, and dependency edges. The hypothesis space is then initialized as $\mathcal{H}^{(1)} = \mathcal{N}(q_c^{(1)})$, the NEAR of this first candidate, comprising all nodes whose answers are not overwritten by correcting $q_c^{(1)}$ and therefore remain

potential error sources. The model then generates a corrected sub-question $q_c^{(1)'}$ and re-executes all nodes in $\mathrm{Desc}(q_c^{(1)})$ under the updated context.

The re-execution outcome serves as a verification signal. If the final answer becomes correct, the correction at $q_c^{(1)}$ is retained and the algorithm terminates. Otherwise, the search continues within the candidate set $\mathcal{H}^{(1)}$. In iteration k, the model identifies a new candidate $q_c^{(k+1)} \in \mathcal{H}^{(k-1)}$, applies a correction, and re-executes $\mathrm{Desc}(q_c^{(k+1)})$. If incorrect, the hypothesis space is contracted as:

$$\mathcal{H}^{(k)} = \mathcal{H}^{(k-1)} \cap \mathcal{N}(q_c^{(k+1)}), \tag{4}$$

excluding $q_c^{(k+1)}$ and the nodes the recorded dependencies place under it, thereby narrowing the candidate set. The algorithm terminates when either a correction recovers the correct answer or $\mathcal{H}^{(k)}$ becomes empty. Since $q_c \notin \mathcal{N}(q_c)$ by definition, each contraction strictly reduces $|\mathcal{H}^{(k)}|$, guaranteeing termination. The complete algorithm is presented in Appendix B.

3.4 Three-Stage Training Framework

We introduce a three-stage training framework that progressively enhances SearchMapper's sub-question decomposition and refinement capabilities.

Stage I: Supervised Fine-Tuning for Imitation. In the first stage, we employ supervised fine-tuning (SFT) to equip SearchMapper with foundational structured reasoning capabilities using high-quality trajectories. During trajectory synthesis, we prompt the model to trigger refinement after addressing critical sub-questions via the special phrase "`Let me make a refinement`", which initiates the summarization and analysis of preceding steps while planning subsequent actions. A critical challenge in SFT lies in ensuring trajectory quality. While we seek trajectories with correct final answers, answer correctness alone does not guarantee high-quality intermediate reasoning processes. To address this, we leverage the scoring model to iteratively refine each sub-question until the quality criteria are satisfied. Only trajectories exhibiting consistently high-quality intermediate steps are retained for training. The SFT objective is formulated as:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(Q,y)\sim\mathcal{D}} \left[\log P_\theta(y \mid Q)\right], \tag{5}$$

where Q denotes the original question, y represents the response trajectory that follows a multi-turn ReAct structure, and θ denotes the model parameters.

Stage II: Preference Optimization for Alignment. To further enhance the model's refinement capability, we employ Kahneman-Tversky Optimization (KTO) (Ethayarajh et al., 2024), which requires only binary-labeled individual trajectories. For each question, the SFT model generates an initial trajectory, and NEAR error detection identifies the first critical erroneous sub-question for refinement and correction. We construct preference data consisting of desirable trajectories $\mathcal{D}_d$ and undesirable trajectories $\mathcal{D}_u$ from three sources:

(1) General refinement trajectories, where the model generates candidate refinements at the critical error node based on a holistic review of the trace. Correct trajectories are added to $\mathcal{D}_d$, otherwise $\mathcal{D}_u$;

(2) Targeted refinement trajectories, where refinements address specific erroneous sub-questions via post-processing, regardless of their distance from the original error. Correct trajectories are added to $\mathcal{D}_d$, otherwise to $\mathcal{D}_u$;

(3) Non-refinement trajectories that are inherently correct without refinement, directly included in $\mathcal{D}_d$ as positive samples that discourage refinement on sound trajectories.

This design ensures the model learns that refinement benefits reasoning only when redirecting toward correctness. By contrasting trajectories with identical refinement positions but differing outcomes, KTO teaches the model both *when* and *how* to refine effectively within the DAG structure.

We optimize via the KTO objective:

$$\mathcal{L}_{\text{KTO}}(\theta) = \mathbb{E}_{(Q,y)\sim\mathcal{D}} \left[w(y) \cdot v_{\text{KTO}}(Q, y; \theta)\right], \tag{6}$$

where $w(y) = \mathbb{I}[y \in \mathcal{D}_d] - \lambda \cdot \mathbb{I}[y \in \mathcal{D}_u]$ balances desirable and undesirable trajectories with $\lambda = |\mathcal{D}_d|/|\mathcal{D}_u|$, and v_{KTO} measures the implicit reward of y relative to the frozen reference π_{ref}.

Stage III: Reinforcement Learning for Generalization. High scores from the scoring model do not guarantee meaningful progress, since it is trained offline and evaluates each sub-question solely on its own content, with no knowledge of how subsequent retrieval unfolds. The agent can exploit this blind spot by proposing well-formed sub-questions that score highly on all three evaluation dimensions while skipping actual retrieval, accumulating dense rewards without grounding reasoning in real evidence. To address this, we introduce a *layer progress reward* that measures whether each layer of sub-questions genuinely moves the reasoning trajectory closer to the correct answer. Concretely, for sub-questions $\{q_t^0, \ldots, q_t^n\}$ proposed at layer ℓ_t, let $\mathcal{H}_t$ denote the context comprising all resolved sub-questions and their retrieved observations from preceding layers, and let $\mathcal{O}_t = \{o_t^0, \ldots, o_t^n\}$ denote the retrieved observations for all sub-questions in ℓ_t. The layer progress reward is defined as:

$$r_{\text{prog}}(\ell_t) = \log P(y \mid Q, \mathcal{H}_t \cup \ell_t \cup \mathcal{O}_t) - \log P(y \mid Q, \mathcal{H}_t), \tag{7}$$

where $\log P(y \mid Q, \cdot)$ is computed via teacher forcing by appending the ground-truth answer tokens and reading their average log-probability. Since $\mathcal{H}_t \cup \ell_t \cup \mathcal{O}_t$ incorporates all search results triggered by layer ℓ_t, $r_{\text{prog}}(\ell_t)$ captures the joint effect of the layer's decomposition and retrieval on answer-oriented progress. Skipping search cannot inflate this signal, as unanswered queries leave $\mathcal{H}_t$ unchanged and yield near-zero or negative rewards that offset scoring model gains. It further tightens the refinement criterion, requiring that the refined sub-question q_t' improve over q_t on both the scoring model evaluation and the layer progress reward.

We integrate the scoring model evaluation and the *layer progress reward* into the Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO) (Yu et al., 2025) framework. For each question Q, we sample a group of G candidate trajectories $\{\mathcal{H}_T^i\}_{i=1}^G$ under the current policy $\pi_{\theta_{\text{old}}}$. Let $\mathcal{L}^i$ denote the set of layers in trajectory $\mathcal{H}_T^i$. The layer-level reward $R^i(\ell_t)$ for layer $\ell_t \in \mathcal{L}^i$ aggregates the scoring model evaluations of all sub-questions in ℓ_t and the layer progress signal $r_{\text{prog}}(\ell_t)$ as defined in Eq. 7. A binary outcome reward $r_{\text{outcome}} \in \{0, 1\}$ is added at the terminal layer to anchor learning to the final answer y, and a format penalty r_{format} is applied at all layers to discourage malformed outputs. The layer-level advantage is estimated via group-relative normalization:

$$\hat{A}^i(\ell_t) = \frac{R^i(\ell_t) - \text{mean}(\mathcal{R})}{\text{std}(\mathcal{R})}, \tag{8}$$

where $\mathcal{R} = \{R^j(\ell) \mid 1 \le j \le G,\ \ell \in \mathcal{L}^j \cup \{\ell^j_{\text{terminal}}\}\}$ collects all layer-level rewards across the group, including terminal layers. The DAPO objective is:

$$\mathcal{J}(\theta) = \mathbb{E}_{Q,\,\{\mathcal{H}_T^i\}\sim\pi_{\theta_{\text{old}}}} \left[\frac{1}{\sum_{i=1}^G |\mathcal{L}^i|} \sum_{i=1}^{G} \sum_{\ell_t \in \mathcal{L}^i} \min \Big(\rho^i(\ell_t)\, \hat{A}^i(\ell_t),\ \text{clip}\big(\rho^i(\ell_t),\, 1-\varepsilon_{\text{low}},\, 1+\varepsilon_{\text{high}}\big)\, \hat{A}^i(\ell_t) \Big) \right], \tag{9}$$

where $\mathcal{L}^i$ denotes the layer set of the DAG induced by trajectory $\mathcal{H}_T^i$, $\rho^i(\ell_t) = \pi_\theta(\ell_t \mid \mathcal{H}_t, Q)/\pi_{\theta_{\text{old}}}(\ell_t \mid \mathcal{H}_t, Q)$ is the importance sampling ratio conditioned on the preceding layer context $\mathcal{H}_t$, and ε_{low}, $\varepsilon_{\text{high}}$ are the asymmetric clipping bounds introduced by DAPO.

4 Experiments

4.1 Experimental Setup

Dataset. To train SearchMapper, we construct a high-quality dataset from multiple sources, including RedSearcher (Chu et al., 2026) and OpenSeeker (Du et al., 2026). The final dataset contains 5000, 2000, and 1000 samples for the SFT, KTO, and DAPO stages, respectively. Trajectories are synthesized using frontier models such as DeepSeek-V3.2 (DeepSeek-AI, 2025) and Claude 4.6 Sonnet (Anthropic, 2026).

Benchmarks. We evaluate the model on five challenging benchmarks. We strictly follow the official evaluation protocol of each. The benchmarks we use are: BrowseComp (Wei et al., 2025), BrowseComp-ZH (Zhou et al., 2025), GAIA-text-only (Mialon et al., 2023), xBench-DS-2505 (Chen et al., 2025b), and xBench-DS-2510 (Chen et al., 2025b).

Table 1: Comparisons of our SearchMapper with other search agents on five benchmarks. The results of SearchMapper are shown in bold.

Model Name	Size	Browse Comp	Browse Comp-ZH	GAIA text-only	xBench DS-2505	xBench DS-2510
Foundation Model with Tools						
GPT-5	-	54.9	63.0	76.4	77.8	-
Gemini-3.0-Pro	-	59.2	66.8	-	-	53.0
Claude-Opus-4.5	-	67.8	62.4	71.5	-	-
DeepSeek-V3.2	671B	67.6	65.0	75.1	78.0	55.7
Search Agent						
Qwen3-30B-A3B-Thinking	30B	5.7	6.3	37.9	44.0	8.0
WebSailorV2–30B	30B	35.3	44.1	74.1	73.7	-
Tongyi DeepResearch	30B	43.4	46.7	70.9	75.0	55.0
IterResearch	30B	37.3	45.2	72.8	71.0	-
AgentFold	30B	36.2	47.3	67.0	-	-
OpenSeeker	30B	29.5	48.4	-	74.0	-
REDSearcher	30B	42.1	49.8	80.1	-	-
SearchMapper	30B	**44.6**	**56.3**	**78.6**	**81.0**	**59.0**

Baselines. We compare SearchMapper with two groups of strong baselines: tool-augmented foundation models, to test whether frontier models can handle long-horizon retrieval without specialized search-agent training, and dedicated search agents, compared at the 30B scale when available. We also include the untrained backbone Qwen3-30B-A3B-Thinking (Team, 2025) under flat ReAct to isolate the contribution of our method. Full baseline details are deferred to Appendix D.1.

4.2 Main Results

Table 1 presents the performance of SearchMapper compared with foundation models and search agents across five benchmarks. Despite using only 30B parameters, SearchMapper achieves 78.6 on GAIA-text-only and 81.0 on xBench-DS-2505, surpassing all foundation models including GPT-5 and DeepSeek-V3.2. On the newly released xBench-DS-2510, SearchMapper achieves 59.0, the highest among all compared methods. Among same-scale 30B search agents, SearchMapper achieves highly competitive performance across GAIA-text-only, xBench-DS-2505, and xBench-DS-2510.

The advantage is most pronounced on xBench-DS benchmarks, where complex questions require resolving interdependent information needs across many retrieval steps, precisely the setting where DAG-structured decomposition and adaptive refinement outperform flat sequential search. On BrowseComp and BrowseComp-ZH, which emphasize deep retrieval for locating single hard-to-find entities, SearchMapper remains competitive, further validating the generalizability of our framework across diverse retrieval paradigms.

4.3 Ablation Study

As shown in Table 2 , we conduct a systematic ablation study on the three-stage training framework to assess the contribution of each stage and its key mechanisms to the performance of SearchMapper.

For Stage I, under the same teacher data, removing only sub-question decomposition and reverting to flat trajectories drops the average by 4.8, while retaining decomposition but removing the scoring model that filters sub-question quality along three dimensions drops the average by 3.4. Together, these results show that DAG-based structured decomposition and its accompanying quality filter each bring independent and substantial gains to SearchMapper's search capability.

For Stage II, we examine the impact of high-quality refinement data by replacing the full preference dataset with only non-refinement trajectories and erroneous refinements. This variant shows a consistent decline across all benchmarks, particularly on complex benchmarks such as BrowseComp

Table 2: Ablation study results for the two key mechanisms in SearchMapper across all three stages of the training framework, with bold for the best results of each stage.

Method	Browse Comp	Browse Comp-ZH	GAIA text-only	xBench DS-2505	xBench DS-2510	Overall
Stage I: Supervised Fine-Tuning						
SearchMapper	**37.3**	**47.2**	**68.9**	**74.0**	**47.0**	**54.9**
w/o scoring model	35.4	41.3	66.7	71.0	43.0	51.5
w/o sub-question decomposition	34.9	39.2	65.1	70.0	41.0	50.1
Stage II: Preference Optimization						
SearchMapper	**40.2**	**49.7**	**70.9**	**75.0**	**50.0**	**57.2**
w/o high-quality refinement	37.8	47.9	69.9	74.0	46.0	55.1
Stage III: Reinforcement Learning						
SearchMapper	**44.6**	**56.3**	**78.6**	**81.0**	**59.0**	**63.9**
w/o process rewards	41.1	51.8	73.8	77.0	53.0	59.3
w/o layer progress rewards	43.5	55.2	77.7	79.0	58.0	62.7
w/o three-dimensional rewards	42.4	52.8	74.8	78.0	56.0	60.8
w/o KTO stage	43.8	54.2	76.7	80.0	56.0	62.1

and xBench-DS-2510, with drops of 2.4 and 4.0 points, respectively, indicating that incorporating diverse refinement sources, particularly sub-question-specific corrections, enables the model to better learn both when and how to perform effective refinement within the DAG structure.

For Stage III, We ablate process rewards and the KTO stage. Removing process rewards drops the average by 4.6, showing that process rewards provide necessary fine-grained supervision for encouraging genuinely useful sub-questions during reinforcement learning. Further decomposing process rewards, removing only the layer progress reward drops the average by 1.2, while removing only the three-dimensional quality reward drops by 3.1, confirming that scoring-model-based quality supervision is the dominant term and echoing the SFT-stage finding that the scoring model spans both stages as a core component for ensuring sub-question quality. Additionally, skipping KTO and applying DAPO directly to the SFT model drops the average by 1.8, showing that preference optimization serves as an important bridge between SFT and reinforcement learning by aligning refinement behavior and providing a stronger initialization for policy exploration.

4.4 Detailed Analysis

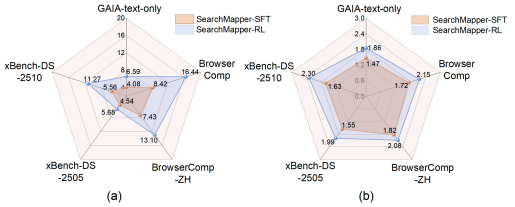


Figure 3: Comparison of SFT and RL models on (a) the average number of sub-questions per question and (b) the average number of tool calls per step. Both results are shown on each of the five benchmarks.

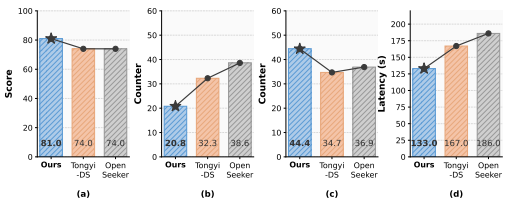


Figure 4: Efficiency–performance comparison of search agents on xBench-DS-2505. (a) Overall performance score; (b) average execution steps per question; (c) average tool calls per question; (d) average end-to-end latency.

Decomposition and Tool-Use Analysis. To understand how RL shapes search behavior, we analyze sub-question decomposition and tool-use density across benchmarks. As shown in Figure 3, the RL model generates more sub-questions per input than the SFT model, with the gap increasing on harder benchmarks: on BrowseComp, RL produces 16.44 sub-questions compared to 8.42 for SFT. This

finer-grained decomposition is accompanied by higher per-step tool-use density, indicating that the model more frequently grounds its reasoning through external retrieval. This pattern is consistent with our DAPO objective, where process rewards favor sub-questions that yield measurable progress toward the answer, encouraging more thorough decomposition and evidence-grounded reasoning.

Efficiency Analysis. To evaluate execution efficiency, we compare average trajectory-level statistics on xBench-DS-2505. As shown in Figure 4, SearchMapper achieves the highest score of 81.0 while requiring only 20.8 reasoning steps on average, 35.6% and 46.1% fewer than Tongyi-DeepResearch and OpenSeeker respectively, directly reducing the most expensive model-side cost. Notably, although SearchMapper issues the most tool calls, its average end-to-end latency remains lower. That's because the DAG structure dispatches independent calls within each layer concurrently, so tool time is bounded by DAG depth and critical-path length rather than total tool calls, avoiding serial accumulation. Overall, the efficiency advantage has two sources: fewer reasoning steps cut the costly model-side latency, while parallel DAG decomposition decouples total tool usage from sequential search depth, yielding a more favorable efficiency-performance tradeoff.

Scoring Model Quality Validation. Since the scoring model is distilled from a stronger teacher via SFT, we validate whether it preserves sufficient scoring fidelity to serve as a process supervisor. We sample 1,000 sub-questions from held-out trajectories and collect binary quality scores from humans, the teacher, and the scoring model. As shown in Figure 5(a), the teacher reaches 92.0% agreement with humans ($\kappa = 0.83$), confirming that it provides reliable supervision for the construction of training data. The scoring model achieves 88.0% with the teacher ($\kappa = 0.78$) and 87.4% with humans ($\kappa = 0.76$), closely matching the teacher with substantially lower computation. All three scorers exhibit conservative bias (more false negatives than false positives), which favors trajectory filtering by preferring to discard borderline sub-questions over admitting low-quality ones. These results support using the scoring model as a reliable process supervisor throughout the training pipeline.

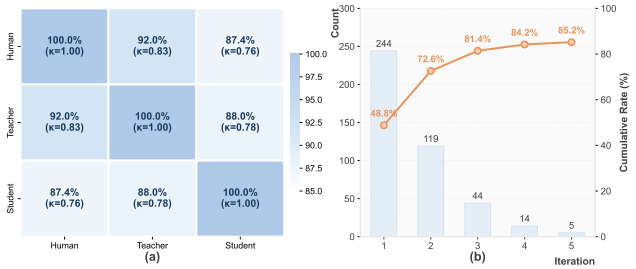


Figure 5: (a) Pairwise agreement among human, teacher-model, and scoring-model quality scores on 1,000 sampled sub-questions, shown by confusion matrices, agreement rates, and Cohen's κ. (b) Cumulative success rate of NEAR-Guided Error Detection over iterations on 500 incorrect trajectories.

NEAR-Guided Error Detection Efficiency. Our algorithm iteratively corrects a suspected node and re-executes its descendants, recording whether the final answer is recovered. A single correction is insufficient: as shown in Figure 5(b), only 48.8% of trajectories are recovered at the first iteration. We apply the procedure to 500 incorrect SFT trajectories and plot the cumulative answer-recovery rate. The curve rises steeply to 72.6% by iteration 2 and then plateaus. Overall, 426 of the 500 trajectories (85.2%) recover a correct answer, with an average of 1.63 iterations among the recovered trajectories. This recovery rate reflects node selection, correction, re-execution, and NEAR contraction together. The remaining 74 trajectories (14.8%) stop before the candidate set is exhausted.

5 Conclusion

We propose SearchMapper, a search-agent framework that addresses the lost in search problem by organizing deep search as a dynamic DAG of dependency-aware sub-questions. This explicit search map enables local sub-question assessment through a hierarchical scoring model and causal error correction through NEAR-guided localization. A three-stage training pipeline further improves decomposition, refinement, and generalization. Experiments on five challenging benchmarks demonstrate strong performance, with ablations verifying the contribution of each proposed component. We hope that structuring search as an explicit, navigable map offers a principled foundation for more reliable and interpretable search agents.

AI USE STATEMENT

Generative AI tools were used for code refinement, grammar checking, and manuscript review. The use of LLMs for trajectory synthesis, scoring, and model training is part of the method, and is described in Sections 3 and 4, and in Appendix D.2. All AI-assisted work was reviewed by the authors, who take responsibility for the final content.

ETHICS STATEMENT

Our experiments use publicly available benchmarks and open-source foundation models, and we follow their licenses and usage policies. The training data are synthesized from publicly accessible sources and do not contain personally identifiable information. We will release the datasets, models, and training scripts under open-source licenses.

REPRODUCIBILITY STATEMENT

Section 3 specifies the DAG formulation, the three dependency-grounded scoring dimensions, NEAR-guided error detection, and the three-stage training pipeline. Section 4 specifies the benchmarks, baselines, and reported results. Appendix A gives the sub-question refinement and selection procedure, Appendix B the NEAR algorithm, and Appendix C the construction of the KTO preference data. Appendix D.2 gives the backbone, retrieval setup, training frameworks, and decoding configuration, and Appendix G provides the system prompt.

REFERENCES

Anthropic. Introducing claude opus 4.5, 2025. URL https://www.anthropic.com/news/claude-opus-4-5.

Anthropic. Introducing claude sonnet 4.6, 2026. URL https://www.anthropic.com/news/claude-sonnet-4-6.

Guoxin Chen, Zile Qiao, Xuanzhong Chen, Donglei Yu, Haotian Xu, Wayne Xin Zhao, Ruihua Song, Wenbiao Yin, Huifeng Yin, Liwen Zhang, et al. Iterresearch: Rethinking long-horizon agents via markovian state reconstruction. *arXiv e-prints*, pp. arXiv–2511, 2025a.

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

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

Zheng Chu, Xiao Wang, Jack Hong, Huiming Fan, Yuqi Huang, Yue Yang, Guohai Xu, Chenxiao Zhao, Cheng Xiang, Shengchao Hu, et al. Redsearcher: A scalable and cost-efficient framework for long-horizon search agents. *arXiv preprint arXiv:2602.14234*, 2026.

Google DeepMind. Gemini 3 pro. https://deepmind.google/models/gemini/pro/, 2025.

DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models, 2025.

Yuwen Du, Rui Ye, Shuo Tang, Xinyu Zhu, Yijun Lu, Yuzhu Cai, and Siheng Chen. Openseeker: Democratizing frontier search agents by fully open-sourcing training data. *arXiv preprint arXiv:2603.15594*, 2026.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*, 2024.

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

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *ArXiv*, abs/2503.09516, 2025. URL https://api.semanticscholar.org/CorpusID:276937772.

Jinzheng Li, Jingshu Zhang, Hongguang Li, and Yiqing Shen. An agent framework for real-time financial information searching with large language models. *arXiv preprint arXiv:2502.15684*, 2024.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Rui Ye, Yida Zhao, Liwen Zhang, Litu Ou, Dingchu Zhang, Xixi Wu, Jialong Wu, et al. Websailor-v2: Bridging the chasm to proprietary agents via synthetic data and scalable reinforcement learning. *arXiv preprint arXiv:2509.13305*, 2025a.

Zixuan Li, Wenxuan Liu, Long Bai, Chunmao Zhang, Wei Li, Fenghui Zhang, Quanxin Jin, Ruoyun He, Zhuo Chen, Zhilei Hu, Fei Wang, Bingbing Xu, Xuhui Jiang, Xiaolong Jin, Jiafeng Guo, and Xueqi Cheng. Knowcoder-v2: Deep knowledge analysis. *arXiv preprint arXiv:2506.06881*, 2025b.

Qiao Liang, Yuke Zhu, Chao Ge, Lei Yang, Ying Shen, Bo Zheng, and Sheng Guo. Learning from the irrecoverable: Error-localized policy optimization for tool-integrated llm reasoning, 2026. URL https://arxiv.org/abs/2602.09598.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023.

Thang Nguyen, Peter Chin, and Yu-Wing Tai. Ma-rag: Multi-agent retrieval-augmented generation via collaborative chain-of-thought reasoning. *arXiv preprint arXiv:2505.20096*, 2025.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*, 2023.

Zile Qiao, Guoxin Chen, Xuanzhong Chen, Donglei Yu, Wenbiao Yin, Xinyu Wang, Zhen Zhang, Baixuan Li, Huifeng Yin, Kuan Li, et al. Webresearcher: Unleashing unbounded reasoning capability in long-horizon agents. *arXiv preprint arXiv:2509.13309*, 2025.

Tianrui Qin, Qianben Chen, Sinuo Wang, He Xing, King Zhu, He Zhu, Dingfeng Shi, Xinxin Liu, Ge Zhang, Jiaheng Liu, Yuchen Eleanor Jiang, Xitong Gao, and Wangchunshu Zhou. Flash-searcher: Fast and effective web agents via dag-based parallel execution. *arXiv preprint arXiv:2509.25301*, 2025.

Ankur Samanta, Akshayaa Magesh, Ayush Jain, Kavosh Asadi, Youliang Yu, Daniel Jiang, Boris Vidolov, Kaveh Hassani, Paul Sajda, Jalaj Bhandari, and Yonathan Efroni. Structure enables effective self-localization of errors in llms. *arXiv preprint arXiv:2602.02416*, 2026.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Huatong Song, Jinhao Jiang, Wenqing Tian, Zhipeng Chen, Yuhuan Wu, Jiahao Zhao, Yingqian Min, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. Smart-searcher: Incentivizing the dynamic knowledge acquisition of llms via reinforcement learning. *arXiv preprint arXiv:2025.findings-emnlp.731*, 2025.

MiroMind Team, S Bai, L Bing, L Lei, R Li, X Li, X Lin, E Min, L Su, B Wang, et al. Mirothinker-1.7 & h1: Towards heavy-duty research agents via verification. *arXiv preprint arXiv:2603.15726*, 2026.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

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

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

Teng Xiao, Yige Yuan, Hamish Ivison, Huaisheng Zhu, Faeze Brahman, Nathan Lambert, Pradeep Dasigi, Noah A. Smith, and Hannaneh Hajishirzi. Meta-reinforcement learning with self-reflection for agentic search. *arXiv preprint arXiv:2603.11327*, 2026.

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

Weinan Zhang, Junwei Liao, Ning Li, Kounianhua Du, and Jianghao Lin. Agentic information retrieval. *arXiv preprint arXiv:2410.09713*, 2025.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, Wenmeng Zhou, and Yingda Chen. Swift:a scalable lightweight infrastructure for fine-tuning, 2024. URL https://arxiv.org/abs/2408.05517.

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

A Sub-question Refinement and Selection

For a sub-question q_t generated at step t, if its overall score satisfies $S_t = 1$, it is considered high-quality and accepted. Otherwise, the sub-question is iteratively refined based on the dimension-wise textual feedback provided by the scoring model:

$$f_{\text{LLM}} : (Q, H_t, q_t^{(k-1)}, G_t^{(k-1)}) \mapsto q_t^{(k)}, k = 1, 2, 3, \tag{10}$$

where $q_t^{(0)} = q_t$ denotes the initial sub-question, and $G_t^{(k-1)} = \{g_t^{\text{red},(k-1)}, g_t^{\text{clar},(k-1)}, g_t^{\text{prog},(k-1)}\}$ represents the textual feedback from the previous evaluation round. The refinement process is repeated for at most three rounds. After each round, the refined candidate is re-evaluated by f_{score}.

If no candidate achieves $S_t^{(k)} = 1$ after the final refinement round, we select the best candidate using a weighted score:

$$W_t^{(k)} = \alpha \cdot s_{t,\text{red}}^{(k)} + \beta \cdot s_{t,\text{clar}}^{(k)} + \gamma \cdot s_{t,\text{prog}}^{(k)}, \tag{11}$$

where α, β, and γ are the weights assigned to redundancy, clarity, and progressiveness, respectively. The final sub-question is selected as:

$$q_t^* = \arg \max_{k \in \{0,1,2,3\}} W_t^{(k)}. \tag{12}$$

To train the scoring model, we use supervised data distilled from Claude-4.6-Sonnet. The data consists of candidate sub-questions generated during problem decomposition, together with quality judgments and textual guidance provided by the teacher model.

We train the scoring model using SFT, enabling it to learn the evaluation preferences of the teacher model. After training, the model can provide quality assessment and feedback signals for candidate sub-questions during inference, thereby supporting the subsequent refinement and selection process.

B NEAR-Guided Error Detection Algorithm

Algorithm 1 NEAR-Guided Error Detection

Require: Incorrect trajectory $\mathcal{H}_T$, DAG $G = (V, E)$, maximum iterations $K_{\max}$
Ensure: A corrected trajectory, or a report of co-occurring errors

Inspect $\mathcal{H}_T$ and identify the most suspicious node $q_c^{(1)}$
$\mathcal{H}^{(1)} \leftarrow \mathcal{N}(q_c^{(1)})$
Correct $q_c^{(1)}$ and re-execute all nodes in $\text{Desc}(q_c^{(1)})$
if the final answer is correct **then**
  return the corrected trajectory
end if
$k \leftarrow 2$
while $\mathcal{H}^{(k-1)} \neq \emptyset$ and $k \leq K_{\max}$ **do**
  Select the most suspicious $q_c^{(k)} \in \mathcal{H}^{(k-1)}$
  Correct $q_c^{(k)}$ and re-execute all nodes in $\text{Desc}(q_c^{(k)})$
  if the final answer is correct **then**
    return the corrected trajectory
  end if
  $\mathcal{H}^{(k)} \leftarrow \mathcal{H}^{(k-1)} \cap \mathcal{N}(q_c^{(k)})$
  $k \leftarrow k + 1$
end while
return a report of co-occurring errors across independent branches

CONVERGENCE GUARANTEE

Theorem B.1 (Finite-Step Termination). *Algorithm 1 terminates within at most $|V| - 1$ iterations.*

Proof. We first establish a key lemma.

Lemma B.2 (Strict Monotone Contraction). *For every iteration $k \geq 1$, $|\mathcal{H}^{(k)}| \leq |\mathcal{H}^{(k-1)}| - 1$.*

Proof of Lemma B.2. By definition of NEAR, $q_c^{(k)} \notin \mathcal{N}(q_c^{(k)})$. Since $q_c^{(k)}$ is selected from $\mathcal{H}^{(k-1)}$, we have $q_c^{(k)} \in \mathcal{H}^{(k-1)} \setminus \mathcal{N}(q_c^{(k)})$, which implies:

$$\mathcal{H}^{(k)} = \mathcal{H}^{(k-1)} \cap \mathcal{N}(q_c^{(k)}) \subsetneq \mathcal{H}^{(k-1)}.$$

Hence $|\mathcal{H}^{(k)}| \leq |\mathcal{H}^{(k-1)}| - 1$. □

We now prove the main theorem. The initial hypothesis space satisfies $|\mathcal{H}^{(1)}| \leq |V| - 1$, since $q_c^{(1)} \notin \mathcal{N}(q_c^{(1)})$ implies $\mathcal{N}(q_c^{(1)}) \subsetneq V$. By Lemma B.2, the hypothesis space size decreases by at least one per iteration, so after at most $|V| - 1$ iterations, $|\mathcal{H}^{(k)}| = 0$, triggering termination. If a correction recovers the correct answer at any earlier iteration, the algorithm terminates immediately. In either case, termination is guaranteed within $|V| - 1$ iterations. □

Remark B.3. The empirically observed average of 1.63 iterations is far below the $|V| - 1$ worst-case bound, as the model's trajectory-level diagnosis identifies the most suspicious node at each step rather than searching exhaustively, achieving substantially faster convergence in practice.

C KTO PREFERENCE DATA CONSTRUCTION

This appendix details the construction of the three-source preference dataset used in Stage II (Section 3.4). Each sample is a pair $(Q, \mathcal{H})$ with a binary label $l \in \{+1, -1\}$ indicating membership in $\mathcal{D}_d$ or $\mathcal{D}_u$, respectively. For each question Q, the SFT model first generates an initial trajectory; the NEAR error detection algorithm (Appendix B) then identifies the first critical erroneous sub-question q_c^* and returns the correction trajectory set $\mathcal{T} = \{(q_c^{(i)}, \mathcal{H}_{\text{corr}}^{(i)}, r^{(i)})\}$.

SOURCE 1: GENERAL REFINEMENT TRAJECTORIES

For the identified critical error node q_c^*, the model is prompted to perform a holistic review of the full reasoning trace $\mathcal{H}_T$ and generate a candidate refinement at the error position. Concretely, the model receives the complete trajectory as context and produces a revised sub-question $q_c^{*\prime}$ based on an overall assessment of where and how the reasoning deviated. The downstream nodes $\text{Desc}(q_c^*)$ are then re-executed under the updated context to obtain the post-refinement trajectory $\mathcal{H}_{\text{gen}}$.

The labeling rule is:

$$\mathcal{H}_{\text{gen}} \in \begin{cases} \mathcal{D}_d & \text{if final answer is correct,} \\ \mathcal{D}_u & \text{otherwise.} \end{cases} \tag{13}$$

Trajectories reaching correct answers under general refinement demonstrate that holistic self-review can successfully redirect the reasoning trajectory; failed trajectories expose cases where surface-level diagnosis is insufficient.

SOURCE 2: TARGETED REFINEMENT TRAJECTORIES

Beyond general refinement, we construct targeted refinement trajectories via post-processing, where refinements directly address specific erroneous sub-questions identified by the NEAR algorithm across multiple iterations. Unlike Source 1, these corrections are not restricted to the first identified error node q_c^*; instead, all candidate nodes $q_c^{(i)}$ surfaced during iterative contraction are eligible, regardless of their depth or distance from the original error position.

For each candidate $q_c^{(i)} \in \mathcal{T}$, a targeted correction is applied by replacing $q_c^{(i)}$ with $q_c^{(i)\prime}$ and re-executing $\mathrm{Desc}(q_c^{(i)})$, yielding $\mathcal{H}_{\mathrm{corr}}^{(i)}$. The labeling rule mirrors Source 1:

$$\mathcal{H}_{\mathrm{corr}}^{(i)} \in \begin{cases} \mathcal{D}_d & \text{if } r^{(i)} = 1, \\ \mathcal{D}_u & \text{if } r^{(i)} = 0. \end{cases} \tag{14}$$

The depth diversity of targeted corrections produces trajectories with refinement positions at varying distances from the terminal node. For illustration purposes, we take a linear sub-question chain $q_1 \to q_2 \to \cdots \to q_n$ as a representative example, where each q_i depends solely on q_{i-1}:

- *Correction close to error origin:* $q_1 \to q_2 \to {q_2}' \to q_3 \to \cdots \to y$ $(\mathcal{D}_d, \checkmark)$
- *Correction further downstream:* $q_1 \to q_2 \to q_3 \to q_4 \to {q_3}' \to \cdots \to y$ $(\mathcal{D}_d, \checkmark)$
- *Targeted correction, wrong node:* $q_1 \to q_2 \to q_3 \to {q_3}' \to \cdots \to \hat{y}$ $(\mathcal{D}_u, \boldsymbol{\times})$
- *Targeted correction, deeper wrong node:* $q_1 \to q_2 \to q_3 \to q_4 \to {q_3}' \to \cdots \to \hat{y}$ $(\mathcal{D}_u, \boldsymbol{\times})$

By contrasting trajectories that share identical refinement positions but differ in outcome (correct y versus incorrect $\hat{y}$), KTO learns both *where* to refine and *how* to refine correctly.

Source 3: Non-Refinement Trajectories

To prevent the model from over-triggering refinement on trajectories that are already progressing correctly, we include a set of non-refinement positive trajectories drawn from the SFT dataset. We randomly sample 500 high-quality trajectories that reach the correct answer without any refinement step, and include them directly in $\mathcal{D}_d$:

$$\mathcal{H}_{\text{no-ref}} = (Q,\ q_1,\ q_2,\ \ldots,\ q_T,\ y) \ \in\ \mathcal{D}_d. \tag{15}$$

To match the turn-length distribution of Sources 1 and 2, trajectories are sampled proportionally across different reasoning lengths. A representative pattern is:

- $q_1 \to q_2 \to q_3 \to q_4 \to q_5 \to y$ $(\mathcal{D}_d, \checkmark)$

These samples encode the behavioral prior that refinement should be withheld when the reasoning trajectory is on track, ensuring that the model learns to condition refinement on *genuine* errors rather than triggering it indiscriminately.

Dataset Statistics

Table 3 summarizes the composition of the final preference dataset. The dataset contains 2,000 trajectories with a desirable-to-undesirable ratio of 3:2.

Table 3: Composition of the KTO preference dataset.

Type	**Label**	**Count**
General refinement (correct)	$\mathcal{D}_d$	250
General refinement (incorrect)	$\mathcal{D}_u$	350
Targeted refinement (correct)	$\mathcal{D}_d$	450
Targeted refinement (incorrect)	$\mathcal{D}_u$	450
Non-refinement (correct)	$\mathcal{D}_d$	500
Total $\mathcal{D}_d$		1,200
Total $\mathcal{D}_u$		800
Total		**2,000**

The data composition reflects three design principles. First, the targeted refinement subset (Source 2) constitutes the largest portion (45%) with a balanced 1:1 positive-to-negative ratio, providing the

strongest contrastive signal for learning effective corrections at specific DAG nodes. Second, the general refinement subset (Source 1) over-represents negative samples (250 vs. 350), reflecting the empirical finding that holistic review without structural guidance more frequently produces ineffective corrections, and encouraging the model to learn that general self-review alone is often insufficient. Third, the non-refinement subset (Source 3) is capped at 500 samples to supply adequate regularization against over-triggering refinement, without dominating the training signal. Overall, 75% of the dataset consists of refinement-related trajectories, directly targeting the alignment objective of Stage II.

D Experimental Setup

D.1 Baselines

We compare SearchMapper with two groups of strong baselines: tool-augmented foundation models and existing search agents. The first group is used to evaluate whether general-purpose frontier models, when equipped with external search tools, can solve long-horizon information-seeking tasks without specialized search-agent training. Specifically, we include GPT-5(Singh et al., 2025), Gemini-3.0-Pro(DeepMind, 2025), Claude-Opus-4.5(Anthropic, 2025), Minimax-M2.1, DeepSeek-V3.2(Liu et al., 2025), and GLM-4.7(Zeng et al., 2025). These models represent recent high-capability foundation models with strong reasoning, instruction-following, long-context understanding, and tool-use abilities.

The second group consists of specialized search agents. Whenever available, we select 30B-scale models to enable a controlled comparison with SearchMapper under comparable model capacity. We include WebResearcher(Qiao et al., 2025), WebSailorV2(Li et al., 2025a), Tongyi DeepResearch(Team et al., 2025), MiroThinker-v1.7-mini(Team et al., 2026), MiroThinker-v1.5(Team et al., 2026), IterResearch(Chen et al., 2025a), AgentFold(Ye et al., 2025), OpenSeeker(Du et al., 2026), and REDSearcher(Chu et al., 2026). These systems cover a diverse range of agent training strategies and search-control paradigms. Since SearchMapper also adopts a 30B backbone, comparisons with this group are particularly informative for isolating the effectiveness of our proposed method. Missing entries in Table 1 indicate that the corresponding model does not report results on the benchmark under the same evaluation setting.

D.2 Implementation Details

We employ Qwen3-30B-A3B-Thinking-2507 (Team, 2025) as the backbone model with a context window of 128k tokens. For web retrieval, we utilize the Serper API to obtain the top 10 document snippets. Training is conducted using the SWIFT (Zhao et al., 2024) and VERL (Feng et al., 2025) frameworks. Both supervised fine-tuning and reinforcement learning are performed on 128 H20 GPUs with Megatron (Shoeybi et al., 2019). We additionally train a student model based on Qwen3-30B-A3B-Thinking-2507 for scoring and sub-question refinement, using training labels annotated by the teacher model Claude 4.6 Sonnet. Following prior work (Team et al., 2026), we evaluate SearchMapper with a maximum of 200 interaction turns. Decoding employs temperature 0.6, top-p 0.95, and a maximum generation length of 8,192 tokens.

E Supplementary Experiments

Table 4: Detailed evaluation results using Avg@3, Pass@1 and Pass@3 metrics.

Metric	GAIA-text-only	BrowseComp-ZH	xBench-DS-2510
Pass@1	78.6	56.3	59.0
Pass@3	84.5	62.5	64.0
Avg@3	79.0	56.7	59.3

Pass@1 and Pass@3 Performance. Since search agents operate in real environments where retrieval results may differ between runs, we report Pass@1, Pass@3, and Avg@3 in Table 4 to separate genuine output diversity from run-level noise. Pass@3 is consistently higher than Pass@1

across all three benchmarks, with an average oracle gain of 5.6 points, indicating that SearchMapper meaningfully produces diverse outputs and benefits from repeated sampling. Meanwhile, Avg@3 remains close to Pass@1, suggesting low variance across samples and stable per-run performance.

F CASE STUDY

The following example illustrates how SearchMapper decomposes a complex multi-hop question into a DAG of progressive sub-questions, performs parallel web retrieval, and iteratively narrows the candidate space until all constraints are satisfied. We present the full reasoning trajectory organized by interaction rounds.

Case Study: Multi-hop Entity Identification via DAG-structured Decomposition

Question: As of the early 2020s, a British broadcaster and journalist, who shares a sibling connection, was educated at the same prominent London institution as the celebrated English aviator who won a major international air race in the 1930s. This journalist was born in a historic university city that shares its name with a famous English county, but their birthplace is not the city where the aviator's great-uncle served as a Unionist politician and judge. The London school they both attended is a medieval foundation formally re-established during the Tudor period, located in a historic county now largely part of a major metropolitan area. Who is this individual?

Round 1 — Clue Analysis & Sub-question Decomposition

SearchMapper first analyzes the dependency structure of the question and identifies the foundational clue: the identity of the aviator is independent and must be resolved first. It constructs the following DAG:

- **Clue A** (independent): A celebrated English aviator who won a major international air race in the 1930s.
- **Clue B** (depends on A): The aviator's great-uncle served as a Unionist politician and judge.
- **Clue C** (depends on A): A London school — medieval foundation, Tudor re-establishment — attended by both the aviator and the journalist.
- **Clue D** (depends on B, C): The journalist was born in a university city named after an English county, excluding the great-uncle's city.

Sub-question 1: *Which celebrated English aviator won a major international air race in the 1930s?*

Search & Answer: Retrieved information about the MacRobertson Trophy Air Race (London to Melbourne, October 1934). The winners were **C.W.A. Scott** (Charles William Anderson Scott) and Tom Campbell Black. Further retrieval confirmed that C.W.A. Scott was educated at **Westminster School** in London — a medieval foundation formally re-established by Queen Elizabeth I in 1560, located in the historic county of Middlesex (now part of Greater London). ✓ All school-related constraints satisfied.

Round 2 — Parallel Sub-question Resolution

Having identified the aviator, SearchMapper recognizes that the next two sub-questions are *independent* of each other and issues them in parallel:

Sub-question 2: *Who was C.W.A. Scott's great-uncle who served as a Unionist politician and judge?*
Sub-question 3: *Which notable British broadcasters or journalists attended Westminster School?*

Search & Answer (Q2): Retrieved that C.W.A. Scott's great-uncle was **Lord Scott-Dickson** (Charles Scott Dickson, 1850–1922), a Scottish Unionist politician associated with **Glasgow** and later **Lord Justice Clerk** based in Edinburgh.

Search & Answer (Q3): The Westminster School alumni list yielded several journalist candidates: **Afshin Rattansi** (born 1968, novelist and journalist), **Adam Boulton** (born 1959, journalist), **Marcel Theroux** (born 1968, novelist and broadcaster), **Giles Coren** (journalist), and others.

Round 3 — Initial Candidate Filtering & Self-Refinement

SearchMapper initially focuses on the two most recognizable candidates — Giles Coren and Marcel Theroux — both of whom have famous siblings (Victoria Coren Mitchell and Louis Theroux, respectively). It applies the birthplace constraint in parallel:

Sub-question 4: *In which city did Lord Scott-Dickson serve as a Unionist politician and judge?*
Answer: Lord Scott-Dickson served as a Unionist politician associated with **Glasgow** and as Lord Justice Clerk based in **Edinburgh**. ✓

Sub-question 5: *Where was Giles Coren born?*
Answer: Born in **Paddington, London** — not a historic university city sharing its name with an English county. × *Eliminated.*

Sub-question 6: *Where was Marcel Theroux born?*
Answer: Born in **Kampala, Uganda** — not a historic university city sharing its name with an English county. × *Eliminated.*

↻ **Sub-question Refinement:** With both initial candidates eliminated, SearchMapper performs a *self-refinement* step. It re-examines the Round 2 alumni search results and identifies a critical oversight: **Afshin Rattansi** (born 1968, journalist) had appeared in the Westminster School alumni list but was not included in the initial candidate evaluation. The model explicitly acknowledges this omission:

> *"I realize I've been overlooking a critical piece of information that appeared in the earlier Westminster School alumni search… Afshin Rattansi (born 1968), listed as a novelist and journalist. This was a significant omission in my candidate evaluation. I narrowed my focus too quickly to familiar names like Giles Coren, Marcel Theroux, and Adam Boulton, without systematically checking all candidates."*

The model then refines its search strategy to include the previously overlooked candidate, issuing new sub-questions to verify Afshin Rattansi's birthplace and sibling connection.

Round 4 — Post-Refinement Candidate Verification

Following the refinement, SearchMapper investigates the recovered candidate:

Sub-question 7: *Where was Afshin Rattansi born, and does he have a notable sibling?*
Answer: Born in **Cambridge**, England — a historic university city sharing its name with the county of Cambridgeshire, and neither Glasgow nor Edinburgh. ✓

Round 5 — Full Constraint Verification & Final Answer

SearchMapper performs a comprehensive verification of the sole surviving candidate against all original constraints:

Sub-question 8: *What is Afshin Rattansi's confirmed career activity as a British broadcaster and journalist in the early 2020s, and does he have a notable sibling connection?*

Search & Answer: Confirmed that Afshin Rattansi presents *Going Underground* (broadcast globally on RT and other platforms) and founded Ghaf TV Productions in the UAE in 2022. His brother **Shihab Rattansi** is a lead anchor at **Al Jazeera English** (previously at CNN International) — both brothers are prominent journalists, constituting the "sibling connection."

Final Verification — All Clues Resolved:

1. **Aviator**: C.W.A. Scott, winner of the 1934 MacRobertson Air Race. ✓
2. **Great-uncle**: Lord Scott-Dickson, Unionist politician (Glasgow) and judge (Edinburgh). ✓
3. **School**: Westminster School — medieval foundation, Tudor re-establishment (1560), historic Middlesex $\subset$ Greater London. ✓
4. **Birthplace**: Cambridge — historic university city, county of Cambridgeshire, $\notin$ {Glasgow, Edinburgh}. ✓

5. **Sibling connection**: Brother Shihab Rattansi, Al Jazeera English lead anchor. ✓
6. **Active in early 2020s**: Presents *Going Underground*, founded Ghaf TV (2022). ✓

Final Answer: Afshin Rattansi

Analysis. This example highlights four key capabilities of SearchMapper: (1) **DAG-structured decomposition** — the model identifies that Clues B and C are independent given Clue A, enabling parallel retrieval in Round 2; (2) **progressive constraint filtering** — Round 3 systematically eliminates candidates (Giles Coren, Marcel Theroux) by applying the birthplace constraint, rather than exhaustively verifying all attributes for every candidate; (3) **sub-question refinement** — when the initial candidate set is exhausted in Round 3, the model does not simply fail but instead performs a self-reflective refinement: it re-examines earlier retrieval results, identifies a previously overlooked candidate (Afshin Rattansi), and explicitly acknowledges the reasoning error that caused the omission (premature focus on familiar names). This demonstrates the model's ability to recover from incomplete candidate evaluation through introspective revision of its own decomposition strategy; and (4) **self-verification** — Round 5 performs a comprehensive confirmation of the sole surviving candidate against all six original constraints before committing to the final answer. The entire trajectory completes in 5 rounds with 8 sub-questions, demonstrating robust multi-hop reasoning with error recovery over a question requiring 6 interlocking constraints.

G PROMPT

System Prompt for SearchMapper

You are a tool-augmented deep search assistant. Your job is leveraging appropriate tools to answer the user's questions. For each question, you must recursively decompose it into a hierarchy of sub-questions with clearly identified dependencies between them, then iteratively call tools to answer each layer of sub-questions from credible and diverse sources. For each step, synthesize the findings into an accurate and well-structured response, until the original question is fully resolved.

Operating Workflow

1. First Response

1.1 Analyze reasoning structure
Analyze each clue in the question one by one, determine the overall reasoning structure, namely Chain, DAG, or Hybrid.

1.2 Declare first-layer sub-questions and dependencies
Based on the reasoning above, finalize the first-layer sub-questions. Parallelize truly independent sub-questions; place dependent ones in subsequent layers. Wrap each sub-question in <sub_question_N>...</sub_question_N> and immediately call tools to search. Indices increment globally and are never reset. Then output a <dependency>...</dependency> tag using original_question as the source.

2. Each Subsequent Response
Each time you receive results from a tool, you must strictly follow these steps.

2.1 Answer the previous layer's sub-questions
Based on tool results, answer each sub-question from the previous layer. Wrap each answer in <sub_question_answer_N>...</sub_question_answer_N> with the core answer in \boxed{}.

2.2 Refine the history and plan the next step
Refine the information gathered so far. Especially when you obtain an important clue, conduct a thorough reflection: recheck the previous logic, organize your current thinking and progress, and analyze what still needs to be accomplished. Output a transition phrase such as "Let me make a refinement." to indicate that reflection is about to begin.

Explicitly reason through the dependency relationships of potential next-layer sub-questions. When proposing dependent sub-questions, annotate each with the specific answer values they depend on, for example:
sub_question_5 depends on Q1 “specific answer value” and Q3 “specific answer value”: sub-question content.
sub_question_6 depends on Q2 “specific answer value” and Q4 “specific answer value”: sub-question content.
Then reflect on whether the proposed sub-questions are necessary and sufficient to resolve the current layer, verify that no redundant or circular dependencies exist, and assess whether the decomposition is reasonable.
Each sub-question must be a complete, independently answerable interrogative sentence. Questions must be clear and unambiguous, avoiding vague references or multiple implicit questions within one. Sub-questions should contain only one question, one question mark, and one interrogative word, such as where, what, which, or when.

2.3 Declare next-layer sub-questions and dependencies
Finalize the next layer of sub-questions with inline dependency annotations. Wrap each sub-question in <sub_question_N>...</sub_question_N>. After proposing each layer of sub-questions, output a <dependency>...</dependency> tag declaring their dependency relationships. Use the following symbols:
A ∥ B: A and B are independent and can be executed in parallel.
A $\rightarrow$ B: B depends on A and can only be posed after A is answered.
A, B $\rightarrow$ C: C depends on both A and B and can only be posed after both A and B are answered.
original_question: used as the source for first-layer sub-questions only.
final_answer: used as the target for the last layer only.
Dependency tag example:
<dependency>
sub_question_1, sub_question_3 $\rightarrow$ sub_question_5
sub_question_2, sub_question_4 $\rightarrow$ sub_question_6
sub_question_5 ∥ sub_question_6
</dependency>

2.4 Deliver the final answer
When the original question can be conclusively resolved, wrap the complete response in <answer>...</answer> with the core answer in \boxed{}.
Tools
You may call one or more functions to assist with the user query. You are provided with function signatures within <tools>...</tools> XML tags.
Search
Performs batched web searches. Supply an array named query; the tool retrieves the top 10 results for each query in one call. The parameter object contains query, which is an array of strings. Include multiple complementary search queries in a single call.
Visit
Parses webpage or webpages and returns a summary of the content according to the goal. The parameters are url and goal. The url field can be a single URL or an array of URLs. The goal field describes the goal of the visit.
PythonInterpreter
Executes Python code in a sandboxed environment. To use this tool, the arguments JSON object must be empty. The Python code to be executed must be placed immediately after the JSON block, enclosed within <code> and </code> tags. Any output that should be seen must be printed to standard output using the print function.
For each function call, return a JSON object with function name and arguments within <tool_call>...</tool_call> XML tags.

Scoring Model Prompt for Sub-question Evaluation

Please independently evaluate each sub-question in the final round of the reasoning trajectory along the following three dimensions. Each dimension should be scored as 0 or 1, where 1 means pass and 0 means fail.

Dimension 1: Redundancy Judgment (Redundancy) [0/1]
Check whether the sub-question contains redundancy.

Cross-level redundancy
Determine whether the sub-question is semantically repeated with any query that has already been raised or answered earlier.
Even if the wording differs, sub-questions with the same query intent should be considered redundant.
Information whose answer has already been obtained through tool calls should not be raised again as a sub-question.

Same-level redundancy
Determine whether the sub-question is redundant with other sub-questions at the same level.
Sub-questions at the same level should not have logical dependencies, meaning that the answer to A should not be a prerequisite for B.
Sub-questions at the same level should not have inclusion relationships in query scope, meaning that A should not be a subset or superset of B.

Scoring criteria
1 (pass): The sub-question has no redundancy, or only negligible wording overlap. It is independent of previous queries and same-level sub-questions, and provides additional information.
0 (fail): The sub-question contains clear redundancy. Its query intent repeats an existing query, it has a dependency or inclusion relationship with a same-level sub-question, or it contributes almost no additional information.

Dimension 2: Clarity Judgment (Clarity) [0/1]
Identify whether the sub-question contains ambiguous expressions from the following three aspects.

Referential clarity
Check whether the sub-question contains unclear references.
For pronouns such as “it”, “the project”, or “the above result”, determine whether they can be clearly resolved from the context.
Check whether the sub-question can be understood independently from the context, that is, whether it makes sense as a standalone search query.

Single intent
Check whether the sub-question implicitly contains multiple independent questions.
Each sub-question should correspond to only one explicit query intent.
If a sub-question actually requires separately querying information about two different entities or dimensions, it should be split into multiple sub-questions.

Explicit query intent
Check whether the sub-question has a clear and executable query intent.
The sub-question should be directly convertible into search keywords or tool-call parameters.
Overly abstract or philosophical sub-questions, such as “What does this mean?”, lack an explicit query intent.

Scoring criteria
1 (pass): The sub-question is clearly expressed, has unambiguous references, contains only one independent query intent, and can be directly converted into a tool call.
0 (fail): The sub-question contains ambiguous references, implies multiple query intents, or has an unclear or non-executable query intent.

Dimension 3: Reasoning Progress Evaluation (Reasoning Progress) [0/1]
Measure whether the sub-question effectively reduces the uncertainty of the original problem and pushes the reasoning toward convergence on the final answer.

Uncertainty reduction

Determine whether the answer to the sub-question substantially narrows the answer space.
Check whether the answer provides key constraints for subsequent reasoning.
Determine whether the sub-question is an idle sub-question, meaning that its answer makes no substantial contribution to the final reasoning.

Necessity: Determine whether the sub-question is necessary for answering the original problem.
Check whether removing the sub-question would cause key information to be missing.
Check whether the sub-question covers a key aspect of the original problem.

Convergence efficiency
Determine whether the sub-question efficiently pushes the reasoning toward the final answer.
Check whether there is a more direct way to obtain the same information.
Check whether the sub-question is an unnecessary detour.

Scoring criteria
1 (pass): The sub-question effectively reduces uncertainty, is a necessary step for answering the original problem, and follows an efficient reasoning path.
0 (fail): The sub-question makes no substantial contribution to the reasoning, is idle or an unnecessary detour, or has an obviously more efficient alternative.