

P-Search: A Parallel Plan–Act–Verify Framework with Decoupled GRPO for Deep Research

Pin Zheng^{1*} Yibo Xue^{1*} Ming Wang^{2†} Yukun Kang¹ Haonan Zheng²
Chengyu Wang² Jian Peng¹ Jun Huang² Changliang Xu^{1†}

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

{zhengpin24, xueyibo24, kangyukun23}@mails.ucas.ac.cn

{pengj, xuchangliang}@ucas.ac.cn

{jinpu.wm, yizhen.zhn, chengyu.wcy, huangjun.hj}@alibaba-inc.com

Abstract

Open-web deep research is limited not only by how long an agent searches, but also by how it organizes evidence into independent, verifiable branches. Existing systems often scale single-path ReAct trajectories or long-context search, which can deepen one hypothesis while leaving alternative entities, constraints, or falsifying evidence underexplored. We propose P-SEARCH, a plan–act–verify framework whose stages run in sequence: it builds a grounded background state, decomposes the task into independent evidence branches, executes those branches in parallel, and iteratively verifies unresolved constraints before final synthesis. For training, we avoid end-to-end credit assignment over a full multi-stage rollout. Instead, P-SEARCH-REACT learns reusable grounded search-report behavior from raw-question search instances and is reused for branch execution and targeted supplementary search, while P-SEARCH-PLAN learns schema-constrained routing, decomposition, and verification decisions. We supervise this structure with star-shaped subgraph data and role-specific GRPO rewards. Experiments show that instantiating P-SEARCH with open-source backbones such as GLM-5 and Tongyi-DR-30B improves their reported baselines on most open-web benchmarks. The trained 8B P-SEARCH-REACT + P-SEARCH-PLAN system reaches 59.2% on GAIA and 70.0% on Xbench-DeepSearch, while trailing the strongest same-scale trained agent on BrowseComp and BrowseComp-ZH. Ablations confirm the complementary roles of Plan, ReAct, and Verify, as well as the transferability of the trained planner across ReAct backbones.

1 Introduction

Deep-research agents must plan under incomplete information, conduct multi-round search and

reading, reconcile conflicting sources, and produce auditable reports beyond standard retrieval-augmented question answering (Huang et al., 2025; Singh et al., 2025; Li et al., 2025d). Recent systems, including OpenAI Deep Research (OpenAI, 2025b,a), DeepResearcher (Zheng et al., 2025), Tongyi DeepResearch (Tongyi DeepResearch Team et al., 2025), and Kimi Researcher (Moonshot AI, 2025), suggest that the core bottleneck is not browsing more pages, but organizing an open evidence space into reliable routes, reports, and verification decisions.

Most workflows still expand research along a single evolving trajectory (Jin et al., 2025a). ReAct (Yao et al., 2023) can deepen one evidence path, but complex questions often contain multiple candidate entities, competing hypotheses, implicit constraints, and long-tail entry points. A single trajectory may anchor on early evidence; longer contexts preserve more history but do not create independent branches, even when they mitigate lost-in-the-middle failures (Liu et al., 2024). Plan-and-Execute methods, such as HuggingGPT (Shen et al., 2023), decompose tasks before execution, but raw-query plans may propagate wrong assumptions to every downstream subtask.

We propose P-SEARCH, a plan–act–verify framework that treats deep research as evidence organization. P-SEARCH first builds a grounded background state, decomposes the task into independently executable evidence branches, dispatches parallel subagents (Tran et al., 2025), and uses an iterative Judge to test whether the collected evidence closes the original constraints. The stages run in sequence, so *parallel* refers throughout to the concurrent branches at Node 3. This design combines grounded planning with independent branch coverage, reducing both brittle decomposition and single-path anchoring.

*Equal contribution.

†Corresponding authors.

The same structure motivates decoupled training. A final-answer reward over the full multi-stage rollout cannot identify whether success or failure came from querying, reading, decomposition, verification, or synthesis; coupled optimization can therefore dilute component-level credit (Zhang, 2026; Kazemnejad et al., 2024; Xi et al., 2025; Ji et al., 2026). Training a separate model for every execution stage is also inefficient. P-SEARCH instead trains two reusable behavior-level policies: P-SEARCH-REACT for grounded ReAct search reports, and P-SEARCH-PLAN for schema-constrained routing, decomposition, and verification decisions.

Our contributions are as follows:

1. We propose P-SEARCH, a plan–act–verify framework whose sequential pipeline covers background modeling, evidence-branch planning, parallel branch execution by subagents, iterative verification, and final synthesis.
2. Based on Wikipedia, we sample star-shaped subgraphs to construct deep-research data oriented toward task decomposition and parallelism. This data is used for GRPO training that decouples the Plan and ReAct components. One trained P-SEARCH-REACT policy is reused across the three ReAct roles (background modeling, branch execution, supplementary search), and the trained P-SEARCH-PLAN transfers as an external planner across ReAct backbones.
3. Experiments show that P-SEARCH, using open-source models such as GLM-5 and Tongyi-DR-30B, improves the self-reported scores of these models on most benchmarks. The trained 8B system shows a complementary profile against same-scale trained agents: clear gains on GAIA and Xbench-DeepSearch, and slightly lower scores on BrowseComp and BrowseComp-ZH.

2 Related Work

This section reviews two lines of work most relevant to our study: search training for deep-research agents and task synthesis for deep-search settings. **Deep research agents.** Deep research agents formulate multi-round search, web browsing, tool use, and evidence synthesis as a long-horizon decision-making problem (Huang et al., 2025; Xi et al., 2025). Search-R1 trains autonomous

search with outcome rewards (Jin et al., 2025b,a); DeepResearcher trains research agents end-to-end in real web-search environments (Zheng et al., 2025); WebThinker uses iterative online DPO to strengthen reasoning–search alternation (Li et al., 2025c); ASearcher studies large-scale asynchronous RL for search training (Gao et al., 2025); WebSailor improves open-web agents using high-uncertainty tasks and agentic RL (Li et al., 2025b,a); DR Tulu trains deep-research policies with RL from evolving rubrics (Shao et al., 2025); and Kimi Researcher and Tongyi DeepResearch represent system-level agentic training for search, browsing, and long-horizon information seeking (Moonshot AI, 2025; Tongyi DeepResearch Team et al., 2025). A common pattern in these systems is to optimize a full search agent or a complete search trajectory, either through online interaction, trajectory-level preference learning, or system-level post-training. This makes them effective at extending long-horizon search, but leaves the internal division of labor among evidence gathering, branch selection, and verification less explicit. OffSeeker further emphasizes that offline deep-research training is bottlenecked by high-quality SFT/DPO data (Zhou et al., 2026); however, such data is still usually organized around complete serial trajectories. P-SEARCH is complementary: it treats the target workflow as heterogeneous roles and trains search-report and structured-planning behaviors separately, so that rewards can constrain search coverage, report utility, route selection, and parallel independence at the behavior level.

Deep-search task synthesis. Deep-search tasks require models to recover implicit constraints, locate evidence, and synthesize answers on the open web, making high-quality synthetic data especially important compared with conventional multi-hop question answering. WebSailor constructs high-uncertainty questions through structural sampling and information obfuscation (Li et al., 2025b); WebExplorer lets models explore web pages autonomously and increases difficulty through long-to-short query evolution (Liu et al., 2025); InfoAgent builds hard-to-retrieve questions with Wikipedia entity trees, subtree sampling, and fuzzification (Zhang et al., 2025); OffSeeker’s DeepForge generates large-scale QA, SFT trajectories, and DPO pairs (Zhou et al., 2026); and DeepDive uses knowledge-graph random walks to generate hard-to-find questions and adds a redundancy penalty during RL (Lu et al., 2025). These

methods mainly synthesize QA pairs or trajectories for serial single-agent search. P-SEARCH instead synthesizes star-shaped subgraphs for parallel evidence-branch execution, explicitly preserving independent evidence branches, merge strategies, and verification gaps to supervise planners that produce parallelizable, verifiable, and mergeable research plans.

3 P-Search Framework

3.1 Design Motivation and Overall Flow

The framework is motivated by two failure modes of long-horizon web research. First, planning directly from a raw query is brittle: ambiguous entity mentions, temporal constraints, or cross-lingual clues can cause early decomposition errors that are copied into every branch. Second, extending a single ReAct trajectory mainly increases depth; it does not guarantee that alternative candidates or falsifying evidence are explored. P-SEARCH therefore separates background modeling, independent branch planning, parallel evidence collection, and merge-time verification. These stages are sequential, and Node 4 loops back when constraints remain open; parallelism is confined to Node 3, where mutually independent branches run concurrently.

In brief, Node 1 builds the background state used for planning; Node 2 either fast-tracks already closed cases or emits independent subtasks; Node 3 executes these subtasks in parallel; Node 4 judges whether the distributed evidence closes the original constraints and triggers targeted supplementary search when needed; and Node 5 produces the final auditable report. Figure 1 shows the execution flow, and Table 1 summarizes the node roles.

3.2 Node 1: Front-End Background Modeling

Raw deep-search queries often contain implicit entities, cross-lingual clues, temporal constraints, or background assumptions that need to be investigated. Node 1 builds an auditable background state, including confirmed facts, candidate entities, constraint chains, ambiguities, and evidence strength for candidate answers. Except for a small number of simple cases where evidence is already closed, Node 1 does not replace the final answer; instead, it provides grounding context for Node 2.

3.3 Node 2: Evidence-Oriented Decomposition

Node 2 takes the original query and the background report as input and outputs schema-constrained JSON, including a fast-track decision, question analysis, independent subtasks, and a verification strategy. Its central constraint is that each subtask must be executable using only the original query and the background report, without depending on results from other subtasks.

We abstract subtasks into four evidence-seeking operations: directly searching for candidate answers, cross-verifying constraints, falsifying alternative candidates, and enumerating sets or extrema. Together, they form a breadth-first-then-depth plan: first expand evidence entry points, then reduce anchoring bias through verification and falsification.

3.4 Node 3: Independent Parallel Subagents

Node 3 assigns each evidence route to an independent subagent, so that it focuses on local verification rather than directly producing a global conclusion. Each subagent receives the original query, one subtask, necessary background, and a brief view of other subtasks. The background view preserves entities and constraints while weakening Node 1’s candidate-answer narrative, reducing anchoring from the preliminary report. Subagents should not depend on an “A before B” execution order; global constraint merging is handled by the Judge and the final synthesis stage.

3.5 Node 4: Iterative Judge and Evidence Closure

Parallel subagents increase evidence coverage, but they do not guarantee that all constraints are correctly merged. Node 4 therefore acts as an iterative verification controller: it projects distributed reports back onto the original query constraints, judges whether the answer is provable, and if not, supplements only the most critical remaining gap.

Each Judge round consists of structured sufficiency assessment, targeted supplementary search, and verification-state update. Specifically, P-SEARCH-PLAN outputs verified constraints, remaining gaps, and the next supplementary-search target; if evidence is insufficient, P-SEARCH-REACT performs bounded search around the gap; the new evidence is then written into the cumulative verification report for the next round. Thus, supplementary search serves explicit verification goals

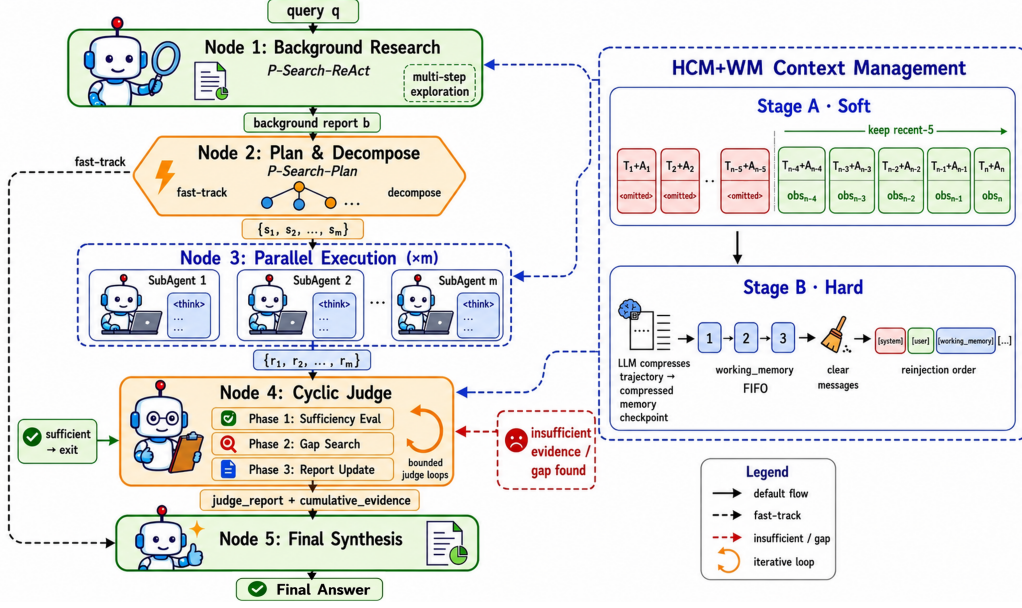


Figure 1: The five-stage execution flow of P-SEARCH. Solid lines denote the default data flow, and dashed lines denote the fast-track bypass. Stages run sequentially (Node 4 may loop back); parallelism applies to the Node 3 branches. HCM+WM is an execution-stability configuration used only by ReAct search stages.

Node	Role in the framework	Main output
Node 1	Preliminary background research and conservative fast-track	Background report, candidate answer, known constraints, and uncertainties
Node 2	Evidence-oriented decomposition	Fast-track decision, analysis, independent subtasks, and merge-time verification strategy
Node 3	Independent parallel subagents	Grounded report and local evidence state for each subtask
Node 4	Iterative Judge and supplementary search	Sufficiency judgment, gap list, supplementary evidence, and cumulative verification report
Node 5	Final synthesis	User-facing final report, answer, and uncertainty statement

Table 1: Node roles and outputs in P-SEARCH. The P-SEARCH-REACT and P-SEARCH-PLAN models trained in this work are one 8B instantiation; larger backbones can also be inserted into the same plan-act-verify workflow.

rather than repeating the full research workflow.

3.6 Node 5: Final Report Synthesis

Node 5 turns separately collected evidence into a readable and auditable final answer. In the full path, it aligns query constraints, subtasks, evidence, and remaining verification gaps. In the fast-track path, it uses the candidate answer and supporting evidence from Node 1 to provide a conclusion and an uncertainty statement.

3.7 Engineering Context-Management Configuration

Long-horizon ReAct accumulates many web observations and may lose candidate states after context resets. To stabilize execution, we adopt an engineering configuration inspired by GLM-5 HCM (GLM-5 Team et al., 2026) and augmented with working memory, denoted by HCM+WM. This

configuration operates only during ReAct search; it does not change the plan-act-verify workflow or the training objectives, and is not treated as an independent algorithmic contribution or performance-attribution target. It keeps the most recent five complete observations and compresses older context into at most three checkpoints when a hard threshold is exceeded.

4 Parallel Data Synthesis and Decoupled GRPO Training

4.1 Star-Shaped Parallel Data Synthesis

Existing web-agent data synthesis often follows a seed-structure expansion-question generation-obfuscation-filtering pipeline, as in WebExplorer’s autonomous web exploration and query evolution, or InfoAgent’s entity tree and fuzzification (Liu et al., 2025; Zhang et al., 2025). These meth-

ods mainly support serial deep search, whereas P-SEARCH needs to supervise planners that produce parallelizable, verifiable, and mergeable research plans.

The key step is **chain- and star-shaped subgraph sampling**. After entity-tree expansion, we sample both linear multi-hop chains and hub-centered star subgraphs. Chain samples provide ordinary multi-hop search supervision, while star-shaped subgraphs select multiple independently verifiable branches around the same hub entity. Each branch corresponds to one constraint, and all branches together determine the final answer. An LLM rewrites the subgraph into a natural-language question, a gold answer, and a canonical plan; entity, numeric, and semantic obfuscation remove direct clues; and multi-round GLM-5 ReAct filtering removes unsolvable, ambiguous, or overly easy samples. Training-set sizes and data-field definitions are deferred to Appendix D.

4.2 Decoupled GRPO Objective

We use GRPO as a standard group-relative policy optimization method: for each fixed prompt, the policy samples a group of candidate outputs, compares them with normalized rewards, and applies a clipped policy update with KL regularization to a reference model (Shao et al., 2024). P-SEARCH does not modify the GRPO update rule; the central design question is how to define comparable prompt groups and behavior-specific rewards for a heterogeneous multi-agent workflow.

This question is nontrivial in a five-node system. End-to-end training with a single outcome reward would dilute credit across free-form search, background reporting, structured decomposition, sufficiency judgment, supplementary search, and final synthesis. Training a separate policy for every node would avoid this dilution but would substantially increase data, training, and serving cost while duplicating shared search-report patterns.

A partial two-node rollout creates a different mismatch. Node 1 GRPO can compare candidate ReAct reports under the same raw research question. Node 2, however, naturally conditions on (q, b) , where b is the upstream background report. If b is generated stochastically by Node 1 inside the same group, planner samples no longer share the same prompt: their rewards may reflect differences in upstream evidence rather than differences in routing, decomposition, or verification quality. This breaks the same-prompt comparison assumption

required by group-relative optimization.

P-SEARCH therefore adopts behavior-level decoupling. P-SEARCH-REACT is trained on Node 1-style raw-question search-report instances and is reused for Node 3 subagent search and Node 4 phase-2 supplementary search through role prompts. P-SEARCH-PLAN is trained on fixed question-background or verification-state inputs and produces schema-constrained JSON for Node 2 planning and Node 4 phase-1 Judge decisions. This compromise keeps reward groups comparable, aligns reward attribution with the behavior being updated, and avoids node-specific policies while still supporting the full five-node execution workflow.

4.3 Reward Design for Search and Planning Behaviors

The reward functions are designed as role-specific verification signals rather than as algorithmic contributions. Both follow the same principle: invalid or shortcut behaviors are handled by multiplicative gates, while an LLM or rule-based score evaluates the quality of the remaining valid output.

Search-report reward. P-SEARCH-REACT is optimized with GRPO on Node 1 raw-question search-report rollouts, and the learned behavior is reused at inference for Node 3 parallel subagents and Node 4 phase-2 supplementary search through role prompts. For a ReAct trajectory τ , we compute a process reward

$$R_{\text{proc}}^{(1)} = V(\tau) (0.5R_{\text{exp}} + 0.5R_{\text{util}}) R_{\text{eff}} R_{\text{dil}}, \quad (1)$$

where V validates tool use and final-report format, R_{exp} measures search breadth, page coverage, and constraint hits, and R_{util} measures the usefulness of the evidence-grounded report. The remaining multiplicative terms penalize redundant or low-yield search and premature termination. The final reward maps this process score into confidence- and correctness-aware regions: sufficiently explored correct answers receive the highest range, correct but under-evidenced answers are capped lower, overconfident wrong answers are discounted, and uncertain answers mainly retain process credit. Appendix E gives the full sub-scores and thresholds.

Planning reward. P-SEARCH-PLAN learns schema-constrained JSON decisions for fast-track, decomposition, and verification planning. Invalid

JSON receives zero reward. Valid outputs enter either a fast-track branch or a decomposition branch:

$$R_{\text{FT}}^{(2)} = C_{\text{route}}, \quad (2)$$

$$R_{\text{DEC}}^{(2)} = C_{\text{route}} C_{\text{dep}} (0.85 R_{\text{pw}} + 0.15 R_{\text{econ}}). \quad (3)$$

Here, C_{route} prevents inappropriate fast-track guesses or overly conservative decomposition, C_{dep} gates plans with obvious sequential dependencies, R_{pw} is an LLM-judge score for plan quality and parallel independence, and R_{econ} discourages excessive or repetitive subtasks. Thus, a high LLM-judge score cannot fully compensate for an invalid route or a non-parallel plan. Appendix F provides the caps, judge inputs, and schema details.

5 Experiments

5.1 Experimental Setup

Evaluation configuration. We evaluate P-SEARCH as a unified deep-research agent equipped with web search, page-reading, and code-execution tools. In the 8B configuration, P-SEARCH-REACT serves Node 1 background modeling, Node 3 parallel subagent search, and Node 4 phase-2 supplementary search; it is also the default model for Node 5 final-answer synthesis. P-SEARCH-PLAN serves Node 2 initial planning and Node 4 phase-1 Judge-style verification. Backbone-transfer experiments keep the same workflow but vary the ReAct backbone or replace only the structured planner with P-SEARCH-PLAN; in these experiments, self-plan means that the backbone itself produces the structured plan inside the P-SEARCH workflow. All P-SEARCH configurations use HCM+WM for execution stability. Detailed step budgets, tool descriptions, context-reset behavior, and slash-separated table entries are deferred to Appendix A. Evaluation uses official metrics or public leaderboard accuracy/exact match where available; internal free-text judgments use GLM-5 as the LLM-as-Judge.

All scores that we compute for P-SEARCH configurations and ablations are fixed-budget **pass@1** results: each question receives one complete system execution and one final answer, with no majority voting, self-consistency, pass@ k resampling, best-run selection, or averaging over multiple random seeds. External baselines are taken from official reports or public leaderboards under their reported accuracy/exact-match protocols. The long-horizon profile in Appendix A changes only

the step budget and context-management variant for BrowseComp and HLE; those scores remain pass@1. As an **inference seed stability check**, two runs of the same checkpoint under different seeds differ by 0.7–1.0 points; we report this as a stability rather than a significance estimate (Appendix A).

Training configuration. Both trainable 8B policies use Qwen3-8B-base as the backbone (Yang et al., 2025). Qwen3 is a competitive open-weight model family with reported strengths in reasoning, multilingual modeling, and agent-oriented tasks, making its 8B base model a suitable starting point for tool-use agent training. We train P-SEARCH-REACT and P-SEARCH-PLAN with decoupled GRPO on two behavior-specific synthetic datasets derived from the same star-shaped construction: one for grounded search-report learning and one for schema-constrained planning and verification decisions. Detailed sample formats, required fields, and hyperparameters are provided in Appendix D.

Benchmarks. We evaluate on five benchmarks. BrowseComp measures persistent English web browsing on difficult short-answer questions (Wei et al., 2025); BrowseComp-ZH measures Chinese multi-hop browsing (Zhou et al., 2025); GAIA evaluates reasoning, browsing, and tool use in general assistants (Mialon et al., 2023); Xbench-DeepSearch evaluates planning, retrieval, reasoning, and summarization in deep search (xbench, 2025; Chen et al., 2025); and Humanity’s Last Exam (HLE) covers expert-level questions across disciplines (Phan et al., 2025). In the main table, BC-en denotes BrowseComp, BC-zh denotes BrowseComp-ZH, and Xbench-DS denotes Xbench-DeepSearch.

5.2 Main Results: Cross-Scale and Cross-Framework Comparison

Table 2 first reports foundation models with tools, followed by trained agents ordered from larger to smaller scales. In the first block, instantiating P-SEARCH with GLM-5 as the ReAct backbone improves BC-en from 62.0/75.9 to 62.7/76.8, BC-zh from 72.7 to 75.1, and HLE from 50.4 to 50.7. The modest HLE gain suggests that the framework mainly improves open-web evidence organization rather than replacing the base model’s expert knowledge. The GLM-5 report does not include GAIA or Xbench-DS; under the P-SEARCH workflow, GLM-5 reaches 80.6 and 82.0 on these two

Method	Model	BC-en	BC-zh	GAIA	Xbench-DS	HLE
Foundation Models with Tools						
Claude-4.5-Opus	Claude-4.5-Opus	67.8	62.4	–	–	43.4
OpenAI-o3	OpenAI-o3	49.7	58.1	70.5	66.7	24.9
OpenAI GPT-5 High	OpenAI GPT-5 High	54.9	65.0	76.4	77.8	35.2
DeepSeek-V3.2	DeepSeek-V3.2	51.4 / 67.6	65.0	–	–	40.8
Kimi-K2.5	Kimi-K2.5	74.9	62.3	–	–	50.2
GLM-5	GLM-5	62.0 / 75.9	72.7	–	–	50.4
P-SEARCH	GLM-5	62.7 / 76.8	75.1	80.6	82.0	50.7
Trained Agents ($\geq 30B$)						
MiroThinker	MiroThinker-v1.0-30B	41.2	47.8	73.5	70.6	–
WebSailor	WebSailor-32B	10.5	25.5	53.2	53.3	–
WebSailor	WebSailor-72B	12.0	30.1	55.4	55.0	–
WebSailor-V2	WebSailor-V2-30B-A3B(SFT)	24.4	28.3	66.0	61.7	–
WebSailor-V2	WebSailor-V2-30B-A3B(RL)	35.4	44.1	74.1	73.7	–
OpenSeeker	OpenSeeker-30B-SFT	29.5	48.4	–	74.0	–
DeepDive	DeepDive-32B-SFT	9.5	23.0	–	48.5	23.9
DeepDive	DeepDive-32B-RL	14.8	25.6	–	50.5	30.6
Tongyi-DR	Tongyi-DR-30B	43.4	46.7	70.9	75.0	32.9
P-SEARCH	Tongyi-DR-30B	44.7	52.6	72.8	77.0	32.5
Trained Agents ($\leq 16B$)						
MiroThinker	MiroThinker-8B-DPO-v0.1	13.0	17.0	57.3	–	–
DeepDive	DeepDive-9B-SFT	5.6	15.7	–	35.0	–
DeepDive	DeepDive-9B-RL	6.3	15.1	–	38.0	–
WebSailor	WebSailor-7B	6.7	14.2	37.9	34.3	–
OffSeeker	OffSeeker-8B-SFT	10.6	24.2	47.6	48.0	–
OffSeeker	OffSeeker-8B-DPO	12.8	26.6	51.5	49.0	–
WebExplorer	WebExplorer-8B-SFT	7.9	21.3	43.7	47.5	–
WebExplorer	WebExplorer-8B-RL	15.7	32.0	50.0	53.7	–
InfoAgent	InfoAgent-14B	15.3	29.2	–	40.4	–
P-SEARCH	P-SEARCH-REACT + P-SEARCH-PLAN	15.3	28.7	59.2	70.0	–

Table 2: Main results across five benchmarks. Slash-separated entries report two budget settings; details are given in Appendix A.

ReAct Backbone	Plan Node	GAIA	Xbench-DS	Note
Qwen3-8B-base	self-plan	48.5	53.0	Untrained backbone and untrained planner
Qwen3-8B-base	P-SEARCH-PLAN	55.3 (+6.8)	56.0 (+3.0)	Plan module improves a base backbone
P-SEARCH-REACT	self-plan	58.3 (+9.8)	68.0 (+15.0)	Only the ReAct search-report behavior is trained
P-SEARCH-REACT	P-SEARCH-PLAN	59.2 (+10.7)	70.0 (+17.0)	Full 8B configuration
Tongyi-DR-30B	self-plan	72.8	77.0	30B backbone with self-planning
Tongyi-DR-30B	P-SEARCH-PLAN	74.7 (+1.9)	78.0 (+1.0)	Plan module still helps a 30B backbone
GLM-5	self-plan	80.6	82.0	Strong tool-use backbone with self-planning
GLM-5	P-SEARCH-PLAN	77.7 (-2.9)	82.0 (+0.0)	Plan module has limited gains on a strong self-planning model

Table 3: Ablation on decoupled training and Plan transfer. The upper block compares an 8B base model, ReAct-only training, Plan-only module, and the full 8B configuration. The lower block evaluates P-SEARCH-PLAN as the structured planner for 30B and GLM-5 backbones.

benchmarks.

In the 30B block, P-SEARCH improves Tongyi-DR-30B from 43.4/46.7/70.9/75.0 to 44.7/52.6/72.8/77.0 on BC-en/BC-zh/GAIA/Xbench-DS, while HLE slightly decreases from 32.9 to 32.5. This again indicates

larger gains in open evidence spaces than in closed expert-level problems.

Among trained agents with no more than 16B parameters, P-SEARCH-REACT + P-SEARCH-PLAN exhibits a complementary cross-benchmark profile rather than uniform superiority. Relative to the

Configuration	Backbone	BC-zh
Only Node 1 (ReAct, max 30 rounds)	Tongyi-DR-30B	42.2
Node 1+2+3 (without verify)	Tongyi-DR-30B	48.1
Node 1+4 (without plan)	Tongyi-DR-30B	48.4
Full P-SEARCH system	Tongyi-DR-30B	52.6

Table 4: Framework-completeness ablation. Experiments are conducted on BrowseComp-ZH, and all configurations use the Tongyi-DR-30B backbone. The full system includes ReAct, Plan, and Verify.

strongest same-scale baseline, WebExplorer-8B-RL, it improves GAIA from 50.0 to 59.2 (+9.2) and Xbench-DS from 53.7 to 70.0 (+16.3), while trailing it slightly on BC-en (15.3 vs. 15.7) and BC-zh (28.7 vs. 32.0). We attribute this profile to training paradigm rather than scale: WebExplorer applies serial-trajectory SFT or distillation before RL, which suits long-tail fact retrieval, whereas our 8B setting uses no teacher-trajectory distillation and trains on star-shaped parallel-plan data with GRPO. A supplemental DR Tulu comparison on WebWalkerQA is given in Appendix A.

5.3 Ablation A: Decoupled Training and Plan Transfer

Table 3 supports the decoupled-training hypothesis. The P-SEARCH +Tongyi-DR-30B and P-SEARCH +GLM-5 entries in Table 2 use the backbone as the planner, so their GAIA/Xbench-DS scores match the corresponding *self-plan* rows here. In the upper block, replacing only the planner of the 8B base backbone with P-SEARCH-PLAN improves GAIA by +6.8 and Xbench-DS by +3.0. Replacing only P-SEARCH-REACT brings a larger Xbench-DS gain, and combining the two yields the best 8B result, showing that planning breadth and evidence depth are complementary. In the lower block, replacing backbone self-planning with P-SEARCH-PLAN gives smaller gains on Tongyi-DR-30B and mostly saturates or slightly regresses on GLM-5, suggesting that P-SEARCH-PLAN is most useful for smaller or less planning-specialized ReAct backbones.

5.4 Ablation B: Framework Completeness

To isolate the contribution of the framework structure itself, we compare four Tongyi-DR-30B configurations on BrowseComp-ZH: long-horizon ReAct only, Plan/subagents without Verify, Verify without Plan/subagents, and the full P-SEARCH system.

Table 4 shows that the full system reaches 52.6

and outperforms all variants missing a key component. Plan/subagents improve coverage but can leave branch omissions or conflicts unresolved, whereas Verify mitigates premature single-path convergence. The full system exceeds the without-verify and without-plan variants by 4.5 and 4.2 points, respectively, indicating that Plan expands independent evidence branches while Verify closes remaining gaps.

5.5 Discussion

Framework gains come from structured budget allocation. Extending a single ReAct trajectory increases the chance of finding evidence, but can also intensify confirmation bias, context pollution, and repeated searches. The gain of P-SEARCH is not equivalent to using more tool calls. Instead, Node 2/Node 3 first expand independent evidence branches, and Node 4 then concentrates additional budget on constraints that remain unclosed. The plan-act-verify structure, with parallel branch execution at Node 3, determines where the budget is spent.

Parallel-plan data versus serial trajectory training. Most same-scale trained agents rely on serial search-trajectory SFT, DPO, or RL. P-SEARCH-PLAN instead learns to decompose a background state into independent evidence branches, merge strategies, and verification targets. This objective better matches the branch-parallel execution of P-SEARCH and helps explain the gains on GAIA and Xbench-DS over several small-scale trained agents. At the same time, P-SEARCH performs no extra SFT on Qwen3-8B and no teacher-trajectory distillation, which explains the relative gap on BrowseComp and BrowseComp-ZH.

Plan transferability and data distribution. P-SEARCH-PLAN consumes only the visible query and background report, and therefore can act as an external planning prior when the downstream ReAct backbone can execute schema-constrained subtasks. This explains the trend in Table 3: the planner gives the largest gain on the untrained 8B backbone, smaller gains after ReAct training or backbone scaling, and does not guarantee a net improvement for GLM-5, whose self-planning behavior is already strong. Thus, planner transfer depends on the match between subtask granularity, search budget, and evidence-merging behavior.

Qualitative case studies. Appendix H provides complete anonymized traces. Here, we use these cases to illustrate three diagnostic patterns: successful independent decomposition, Judge-driven recovery from an unclosed constraint, and interpretable failure caused by misleading background evidence, dependent subtasks, premature Judge termination, or overconfident synthesis.

6 Conclusion

We presented P-SEARCH, a plan-act-verify framework that organizes deep research into a sequential pipeline of background modeling, independent evidence branching, parallel branch execution, and iterative verification. Experiments show that Plan, ReAct, and Verify are complementary; P-SEARCH-PLAN transfers across 8B and 30B backbones; and star-shaped parallel data provides a supervision signal distinct from serial trajectory imitation. These results suggest that improving deep research requires evidence-organization structures and training distributions aligned with parallel evidence-branch execution, rather than only longer ReAct trajectories.

Limitations

The current 8B system still lags behind methods using high-quality teacher trajectories or serial SFT+RL on BrowseComp and BrowseComp-ZH, suggesting that star-shaped parallel data with RL alone does not fully cover long-tail fact retrieval. In the 30B setting, HLE also shows that a search-oriented framework is still bounded by the base model’s expert knowledge and reasoning capability. This work includes key ablations over ReAct, Plan, and Verify, but does not exhaustively cover the number of subagents, Judge strategy, or subgraph-sampling variants; therefore, we do not attribute performance gains to untested low-level implementation details. Free-text evaluation still relies on LLM-as-Judge (Han et al., 2025), and future work should add human review and report agreement. All reported scores are single-run pass@1; the seed and cross-judge checks are stability checks, not significance estimates. Finally, the training distribution of P-SEARCH-PLAN comes from star-shaped synthetic parallel trajectories; more diverse synthesis, SFT initialization, and cross-backbone trajectory collection may further improve transferability.

Ethical Considerations

P-SEARCH targets information retrieval and report generation, but may still inherit errors from search engines, web pages, and LLM judges. In user-facing settings, the system should expose sources, uncertainty, and unresolved gaps. In medical, legal, financial, or safety-critical decisions, it should not be used as the sole basis without expert review.

Acknowledgments

This work was supported by Alibaba Innovative Research Program.

References

- Kaiyuan Chen, Yixin Ren, Yang Liu, Xiaobo Hu, Hao-tong Tian, Tianbao Xie, Fangfu Liu, Haoye Zhang, Hongzhang Liu, Yuan Gong, Chen Sun, Han Hou, Hui Yang, James Pan, Jianan Lou, Jiayi Mao, Jizheng Liu, Jinpeng Li, Kangyi Liu, Kenkun Liu, Rui Wang, Run Li, Tong Niu, Wenlong Zhang, Wenqi Yan, Xu-anzheng Wang, Yuchen Zhang, Yi-Hsin Hung, Yuan Jiang, Zexuan Liu, Zihan Yin, Zijian Ma, and Zhiwen Mo. 2025. [xbench: Tracking agents productivity scaling with profession-aligned real-world evaluations](#). *Computing Research Repository*, arXiv:2506.13651.
- 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](#). *Computing Research Repository*, arXiv:2508.07976.
- GLM-5 Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chengxing Xie, Chenzheng Zhu, et al. 2026. [GLM-5: From vibe coding to agentic engineering](#). *Computing Research Repository*, arXiv:2602.15763. Version 2.
- Steve Han, Gilberto Titericz Junior, Tom Balough, and Wenfei Zhou. 2025. [Judge’s verdict: A comprehensive analysis of LLM judge capability through human agreement](#). *Computing Research Repository*, arXiv:2510.09738.
- Yuxuan Huang, Yihang Chen, Haozheng Zhang, Kang Li, Huichi Zhou, Meng Fang, Linyi Yang, Xiaoguang Li, Lifeng Shang, Songcen Xu, Jianye Hao, Kun Shao, and Jun Wang. 2025. [Deep research agents: A systematic examination and roadmap](#). *Computing Research Repository*, arXiv:2506.18096.
- Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xi-angxiang Chu, and Liaoni Wu. 2026. [Tree search for LLM agent reinforcement learning](#). In *International Conference on Learning Representations*. ICLR 2026.

- Bowen Jin, Jinsung Yoon, Priyanka Kargupta, Sercan O. Arik, and Jiawei Han. 2025a. [An empirical study on reinforcement learning for reasoning-search interleaved LLM agents](#). *Computing Research Repository*, arXiv:2505.15117.
- 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](#). *Computing Research Repository*, arXiv:2503.09516.
- Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordani, Siva Reddy, Aaron Courville, and Nicolas Le Roux. 2024. [VinePPO: Unlocking RL potential for LLM reasoning through refined credit assignment](#). *Computing Research Repository*, arXiv:2410.01679. Accepted at ICML 2025.
- Kuan Li, Zhongwang Zhang, Huifeng Yin, Rui Ye, Yida Zhao, Liwen Zhang, Litu Ou, Dingchu Zhang, Xixi Wu, Jialong Wu, Xinyu Wang, Zile Qiao, Zhen Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025a. [WebSailor-V2: Bridging the chasm to proprietary agents via synthetic data and scalable reinforcement learning](#). *Computing Research Repository*, arXiv:2509.13305.
- Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025b. [WebSailor: Navigating super-human reasoning for web agent](#). *Computing Research Repository*, arXiv:2507.02592.
- Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yutao Zhu, Yongkang Wu, Ji-Rong Wen, and Zhicheng Dou. 2025c. [WebThinker: Empowering large reasoning models with deep research capability](#). *Computing Research Repository*, arXiv:2504.21776. Accepted at NeurIPS 2025.
- Yangning Li, Weizhi Zhang, Yuyao Yang, Wei-Chieh Huang, Yaozu Wu, Junyu Luo, Yuanchen Bei, Henry Peng Zou, Xiao Luo, Yusheng Zhao, Chunkit Chan, Yankai Chen, Zhongfen Deng, Yinghui Li, Haitao Zheng, Dongyuan Li, Renhe Jiang, Ming Zhang, Yangqiu Song, and Philip S. Yu. 2025d. [A survey of RAG-reasoning systems in large language models](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 12120–12145, Suzhou, China. Association for Computational Linguistics.
- Junteng Liu, Yunji Li, Chi Zhang, Jingyang Li, Aili Chen, Ke Ji, Weiyu Cheng, Zijia Wu, Chengyu Du, Qidi Xu, Jiayuan Song, Zhengmao Zhu, Wenhu Chen, Pengyu Zhao, and Junxian He. 2025. [WebExplorer: Explore and evolve for training long-horizon web agents](#). *Computing Research Repository*, arXiv:2509.06501.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. [Lost in the middle: How language models use long contexts](#). *Transactions of the Association for Computational Linguistics*, 12:157–173.
- 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](#). *Computing Research Repository*, arXiv:2509.10446.
- Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. [GAIA: A benchmark for general AI assistants](#). *Computing Research Repository*, arXiv:2311.12983.
- Moonshot AI. 2025. [Kimi-Researcher: End-to-end RL training for emerging agentic capabilities](#). Project page. Accessed: 2026-05-22.
- OpenAI. 2025a. [Deep research system card](#). System card, OpenAI. Accessed: 2026-05-22.
- OpenAI. 2025b. [Introducing deep research](#). OpenAI product blog. Accessed: 2026-05-22.
- Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, Michael Choi, Anish Agrawal, et al. 2025. [Humanity’s last exam](#). *Computing Research Repository*, arXiv:2501.14249.
- Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, Samuel G. Finlayson, David Sontag, et al. 2025. [DR tulu: Reinforcement learning with evolving rubrics for deep research](#). *Computing Research Repository*, arXiv:2511.19399.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. [DeepSeekMath: Pushing the limits of mathematical reasoning in open language models](#). *Computing Research Repository*, arXiv:2402.03300.
- Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. [HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 38154–38180.
- Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025. [Agentic retrieval-augmented generation: A survey on agentic RAG](#). *Computing Research Repository*, arXiv:2501.09136.
- Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, Kuan Li, Liangcai Su, Litu Ou, Liwen Zhang, Pengjun Xie, Rui Ye, Wenbiao Yin, Xinmiao Yu, Xinyu Wang, Xixi Wu, Xuanzhong Chen, Yida Zhao, Zhen Zhang, Zhengwei Tao, Zhongwang Zhang, Zile Qiao, Chenxi Wang, Donglei Yu, Gang Fu, Haiyang Shen, Jiayin Yang, Jun Lin, Junkai Zhang, Kui Zeng, Li Yang,

- Hailong Yin, Maojia Song, Ming Yan, Peng Xia, Qian Xiao, Rui Min, Ruixue Ding, Runnan Fang, Shaowei Chen, Shen Huang, Shihang Wang, Shihao Cai, Weizhou Shen, Xiaobin Wang, Xin Guan, Xinyu Geng, Yingcheng Shi, Yuning Wu, Zhuo Chen, Zijian Li, and Yong Jiang. 2025. [Tongyi DeepResearch technical report](#). *Computing Research Repository*, arXiv:2510.24701.
- Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O’Sullivan, and Hoang D. Nguyen. 2025. [Multi-agent collaboration mechanisms: A survey of LLMs](#). *Computing Research Repository*, arXiv:2501.06322.
- 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](#). *Computing Research Repository*, arXiv:2504.12516.
- Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and Fei Huang. 2025. [Web-walker: Benchmarking LLMs in web traversal](#). *Computing Research Repository*, arXiv:2501.07572.
- xbench. 2025. [xbench-DeepSearch: AI search & tool usage benchmark](#). Benchmark page. Accessed: 2026-05-22.
- Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazhang Zhang, Wenxiang Chen, Wei He, Yiwen Ding, Guanyu Li, Zehui Chen, Zhengyin Du, Xuesong Yao, Yufei Xu, Jiecao Chen, Tao Gui, Zuxuan Wu, Qi Zhang, Xuanjing Huang, and Yu-Gang Jiang. 2025. [AgentGym-RL: Training LLM agents for long-horizon decision making through multi-turn reinforcement learning](#). *Computing Research Repository*, arXiv:2509.08755.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. [Qwen3 technical report](#). *Computing Research Repository*, arXiv:2505.09388.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. [ReAct: Synergizing reasoning and acting in language models](#). In *International Conference on Learning Representations*. ICLR 2023.
- Chenchen Zhang. 2026. [From reasoning to agentic: Credit assignment in reinforcement learning for large language models](#). *Computing Research Repository*, arXiv:2604.09459.
- Gongrui Zhang, Jialiang Zhu, Ruiqi Yang, Kai Qiu, Miaosen Zhang, Zhirong Wu, Qi Dai, Bei Liu, Chong Luo, Zhengyuan Yang, Linjie Li, Lijuan Wang, Weizhu Chen, Yuan Zhang, Xin Li, Zhaoyi Liu, Xin Geng, and Baining Guo. 2025. [InfoAgent: Advancing autonomous information-seeking agents](#). *Computing Research Repository*, arXiv:2509.25189.
- 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, Yuxin Gu, Sixin Hong, Jing Ren, Jian Chen, Chao Liu, and Yining Hua. 2025. [BrowseComp-ZH: Benchmarking web browsing ability of large language models in chinese](#). *Computing Research Repository*, 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](#). *Computing Research Repository*, arXiv:2601.18467.

A Verified Implementation Constants

This appendix documents the implementation constants that are referenced in the main paper but are too granular for the experimental-setup section. All values are taken from the configuration file actually used to produce the reported numbers. We separately report (i) the ReAct step budgets for Node 1 (PRELIMINARY), Node 3 (WORKER) and Node 4 (JUDGE); (ii) the per-iteration tool-call cap; (iii) the maximum number of Judge re-entry rounds; (iv) the context-window threshold mt (in characters) that gates the HCM+WM pipeline; (v) the tool stack and the page-summarization model; and (vi) the LLM-as-Judge model used for offline evaluation.

Two evaluation profiles. We adopt *two* step-budget profiles, chosen according to the typical trajectory length needed to answer the benchmark. The **default profile** is used for GAIA, Xbench-DS and BrowseComp-ZH, where solvable trajectories usually terminate within a few dozen ReAct steps; under this profile every agent (Node 1 /

Node 3 / Node 4) keeps the standard short budget, the Judge is allowed up to five re-entry rounds for residual fact-checking, and the context manager is the HCM+WM configuration described in the main text. The **long-horizon profile** is used for BrowseComp and HLE, on which the agent often has to traverse hundreds of pages before the constraints converge. Under the long-horizon profile every agent receives a step budget of 200; to avoid context overflow we replace HCM+WM with the GLM-5-style HCM+discard_all variant introduced in Appendix B; and, since each Judge re-entry alone may consume tens of search steps, the per-task Judge cap is conservatively reduced to two rounds.

Constant	Default	Long-horizon
PRELIMINARY_MAX_STEPS	30	200
WORKER_MAX_STEPS	25	200
JUDGE_MAX_STEPS	25	200
MAX_TOOL_CALLS_PER_ITER	7	7
MAX_JUDGE_LOOPS	5	2
Context manager	HCM+WM	HCM+discard_all
Benchmarks	GAIA / Xbench-DS / BC-zh	BC-en / HLE

Table 5: ReAct step budgets, per-iteration tool-call cap, Judge re-entry cap, and context-management variant under the two evaluation profiles. The long-horizon profile is used only for BrowseComp and HLE; the remaining three benchmarks use the default profile.

Context threshold mt . The hierarchical context manager is parameterized by a single backbone-dependent character threshold mt . Once the running message buffer exceeds mt , the Stage A keep-recent- K soft compression is applied; if the buffer remains above the hard ceiling $mt \times 1.5$, the Stage B LLM-checkpoint hard reset is triggered (Appendix B). The thresholds we use for the three backbones reported in the paper are listed in Table 6; the unit is raw characters of the assembled message buffer.

Backbone	mt (characters)
GLM-5	100k
Tongyi-DR-30B	55k
P-SEARCH-REACT (8B)	35k

Table 6: Backbone-specific context thresholds mt used by the hierarchical context manager. Smaller backbones operate with smaller mt to leave headroom for working-memory checkpoints within their effective context window.

Tool stack. At evaluation time every agent (Node 1 / Node 3 / Node 4) is given the same three-tool

stack: (1) a **web search** tool backed by Tavily and Google Search, returning a ranked list of URL + snippet results; (2) a **page-reading** tool (visit) that fetches the target URL and *summarizes the cleaned page content with GLM-5* before returning it to the calling agent, so that no agent ever consumes raw HTML; and (3) a **code-execution** tool that runs short Python snippets in a sandbox for arithmetic, unit conversion, list aggregation and other lightweight data manipulation. The same stack is exposed to all three backbones, so that the tool API itself is not a confounding factor across systems.

LLM-as-Judge for offline evaluation. For free-text benchmarks (BrowseComp, BrowseComp-ZH, HLE, GAIA, and Xbench-DS) we score each final report against the gold answer with an LLM judge. The judge model is fixed to glm-5 through the EVAL_JUDGE_MODEL setting and held identical across all backbones and ablations, so that judge-side variance does not interact with the system being evaluated.

Compute and efficiency. Because P-SEARCH replaces a single long trajectory with a short sequential pipeline whose Node 3 branches run concurrently, its cost is best read as tool calls per question rather than wall-clock latency, which is dominated by API queueing. We therefore report tool-call counts instead of a separate efficiency table. On BrowseComp-ZH the GLM-5 configuration issues on average 576 tool calls per question (468 search and 108 visit calls) aggregated over all nodes and subagents, which is the same order of magnitude as the single-path step budgets (~ 580 steps) reported for comparable long-horizon single-agent settings; the parallel branches thus redistribute a comparable tool budget across independent evidence paths rather than enlarging it (accuracy for the same configuration is reported in Table 2). In the GAIA ablations, the trained 8B configuration issues on average 127 tool calls per question (97 search and 30 visit) against 93 calls (64 search and 29 visit) for the untrained Qwen3-8B backbone under the identical pipeline, i.e. training increases search breadth by roughly one third while leaving page-reading cost essentially unchanged. These numbers are averages over the corresponding evaluation runs and are meant as an order-of-magnitude cost characterization, not as a controlled efficiency benchmark: the systems being compared differ in backbone and in the number of active subagents.

Inference seed stability. A complete P-SEARCH execution is expensive, so we do not estimate multi-seed variance for every configuration; instead we re-ran the same checkpoints with a different inference seed on three of the benchmarks. The two runs differ by 0.7–1.0 points, which we report as a stability check on the inference pipeline rather than as a variance or significance estimate (§5.1); judge-side stability is treated separately in Appendix G.

Supplemental comparison with DR Tulu. DR Tulu trains deep-research policies with reinforcement learning from evolving rubrics (Shao et al., 2025) and does not report the five benchmarks of Table 2, so it cannot be placed in that table. On WebWalkerQA (Wu et al., 2025), the one benchmark reported by both systems, DR-Tulu-SFT-8B and DR-Tulu-8B obtain 31.9 and 39.0, whereas the trained 8B P-SEARCH configuration obtains 61.0 under our evaluation harness. Since the two systems differ in tool stack and judge model, we treat this as a supplemental single-benchmark reference point rather than a controlled comparison.

B Context Management and Working-Memory Prompts

The main text introduces HCM+WM only at the level of design intent. This appendix gives the precise two-stage rule that we actually run during long search trajectories: a soft Stage A that drops old tool observations in place, and a hard Stage B that checkpoints the trajectory and resets the message buffer. Both stages share a single backbone-dependent character threshold mt (Appendix A); only the long-horizon profile escalates the configuration to the GLM-5-style HCM+discard_all variant described below.

Stage A: keep-recent- K soft compression with a fixed placeholder. At every ReAct turn we estimate the size S of the assembled message buffer in characters. As soon as S exceeds mt , Stage A is applied: we keep the most recent K tool observations intact (we use $K=5$ throughout the paper), and replace the textual content of every *older* tool observation by a single fixed sentence: “Tool result is omitted to save tokens.” The associated tool-call identifier and tool name are preserved, so that the agent still sees *which* tool was invoked at every past step — only the bulky observation text is discarded. Stage A is therefore a deterministic, in-place rewrite that requires no LLM call, and is

sufficient on the majority of trajectories.

Stage B: LLM checkpoint hard reset. Stage B fires only when Stage A is no longer enough. Let S_{msg} denote the post-Stage-A buffer size excluding working memory, and S_{total} the size including working memory. We trigger Stage B whenever

$$S_{msg} > mt \quad \text{or} \quad S_{total} \geq 1.5 \, mt,$$

i.e. whenever the messages alone still exceed the threshold, or the total assembled context approaches the hard ceiling at $1.5 \, mt$. When Stage B is triggered, the current (Stage-A-compressed) trajectory is summarized by an LLM call into a short checkpoint of at most 2,000 characters and appended to a FIFO working-memory buffer that retains up to three checkpoints; the running message list is then cleared back to [system, user], and on the next turn the working-memory buffer is re-injected immediately after the user message. The agent therefore continues from a fresh context that still carries a distilled summary of its prior research state across the reset.

C Parallel Data Synthesis and Trajectory Collection

This appendix documents the data pipeline that produces the star-shaped, parallelizable training examples used to train P-SEARCH-PLAN. Stages S1–S3, S5 and S6 follow standard practice in deep-research data synthesis (Liu et al., 2025; Zhang et al., 2025), so we sketch them only at the level needed to make the algorithm self-contained, and we concentrate on the two pieces that are specific to P-SEARCH: (i) the parallel question and canonical plan synthesis at stage S4, and (ii) the mock-trajectory solvability filter at stage S6 that simultaneously yields the trajectory-filtered (q , background) pairs consumed by Node 2.

C.1 Six-stage synthesis pipeline

Algorithm 1 gives the full procedure. Given a seed corpus and a target branching factor, the pipeline expands per-seed entity trees, samples star-shaped subgraphs with one hub and several independent branches, synthesizes one parallelizable question and its canonical plan per star (stage S4, prompts in Figures 2 and 3), fuzzifies entity mentions to remove shortcut answers, and finally accepts only those questions whose mock-trajectory rollouts contain at least one verified-correct solution; the winning trajectory’s background report is exported

together with the question and plan as one Node 2 training example.

Algorithm 1 Star-shaped parallel data synthesis (six-stage pipeline).

Require: seed corpus $\mathcal{C}_0$; target branch count B ; fuzzification level f ; mock trials per question N ; mock-visit model M_{mock}

Ensure: trajectory-filtered training set $\mathcal{D} = \{(q, \text{plan}, \text{bg})\}$

```

1:  $\mathcal{D} \leftarrow \emptyset$ 
   // S1: seed mining
2:  $\mathcal{S} \leftarrow \text{MINESEEDS}(\mathcal{C}_0)$ 
   // S2: entity-tree expansion
3:  $\mathcal{T} \leftarrow \{\text{EXPANDTREE}(s) \mid s \in \mathcal{S}\}$ 
   // S3: star-shaped subgraph sampling
4:  $\mathcal{G} \leftarrow \text{SAMPLESTARS}(\mathcal{T}, B) \quad \triangleright \text{each } g \in \mathcal{G}: \text{one hub} + B \text{ branches}$ 
   // S4: parallel question and canonical plan
5:  $\mathcal{Q} \leftarrow \emptyset$ 
6: for each  $g = (h, \text{br}, a) \in \mathcal{G}$  do
7:    $(q, \text{sq}) \leftarrow \text{QSYNTH}_{\parallel}(h, \text{br}, a, B) \quad \triangleright \text{Fig. 2}$ 
8:    $\text{plan} \leftarrow \text{PLANGEN}_{\parallel}(q, a, \text{br}, \text{sq}) \quad \triangleright \text{Fig. 3}$ 
9:    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(q, a, \text{plan})\}$ 
10: end for
   // S5: fuzzification
11:  $\mathcal{Q} \leftarrow \{(\text{FUZZIFY}(q, f), a, \text{plan}) \mid (q, a, \text{plan}) \in \mathcal{Q}\}$ 
   // S6: ReAct trajectory-based solvability filter
12: for each  $(q, a, \text{plan}) \in \mathcal{Q}$  do
13:    $\mathcal{T}_q \leftarrow \{\text{RUNPSEARCH}(q; \text{visit}=M_{\text{mock}})\}_{i=1}^N$ 
14:    $\rho \leftarrow \text{SCOREANDRANK}(\mathcal{T}_q, a)$ 
15:   if  $\max_i \rho_i \geq \text{correct}$  then
16:      $\tau^* \leftarrow \arg \max_{\tau \in \mathcal{T}_q} \rho(\tau)$ 
17:      $\text{bg} \leftarrow \text{EXTRACTBACKGROUND}(\tau^*)$ 
18:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(q, \text{plan}, \text{bg})\}$ 
19:   end if
20: end for
21: return  $\mathcal{D}$ 

```

C.2 Stage S4: parallel question and canonical plan

Stage S4 turns each star-shaped subgraph (h, br, a) into one multi-branch research question and a canonical parallel plan via two prompts, $\text{QSYNTH}_{\parallel}$ and $\text{PLANGEN}_{\parallel}$ (Figures 2 and 3). The former emits a multi-branch question with a fixed merge type; the latter turns that question into a *canonical* parallel plan in which all research sub-tasks have zero dependencies and only a single MERGE task aggregates them. Source code and datasets will be released upon acceptance.

C.3 Stage S6: solvability filter and Node 2 training-pair collection

Stage S6 is implemented as a single rollout-based pass that serves two purposes simultaneously. For each candidate question we run the full P-SEARCH loop N times, with the live `visit` tool replaced by a deterministic GLM-5 mock that summarizes the gold-anchor pages into citation-only excerpts

(RUNPSEARCH with M_{mock}). The N trajectories are scored against the gold answer by a frozen LLM judge (SCOREANDRANK, with a glm-5 fallback chain to keep the filter operational under transient API errors).

A question is retained if at least one of its N trajectories is verified correct; the highest-scoring trajectory τ^* then contributes the $(q, \text{plan}, \text{bg})$ tuple to the training set, where the background report bg is the Node 1 preliminary report extracted from τ^* . Questions with zero correct trajectories under N trials are discarded as unsolvable — empirically, these are dominated by over-fuzzified questions and by questions whose ground-truth evidence falls outside the mock corpus. This single filter therefore both rejects ill-posed entries from the synthetic pool and produces the exact $(q, \text{background})$ pair format that P-SEARCH-PLAN consumes, so no second-pass data construction is required.

D Training Hyperparameters and Data Format

This appendix complements Appendix C. We first contrast the two structural types of synthesized questions with a concrete star-shaped/linear pair (§D.1), then enumerate the two per-policy sample schemas (§D.2), and finally list the GRPO training hyperparameters used to optimize P-SEARCH-REACT and P-SEARCH-PLAN (§D.3).

D.1 Star-shaped vs. linear synthetic questions

The synthesis pipeline in Appendix C produces two structural types of questions. A *star-shaped* question has a hub entity from which several disjoint reasoning branches radiate; each branch is itself a short chain that terminates in an atomic fact, and a final MERGE task aggregates the per-branch answers. A *linear* question is a single multi-hop chain in which every hop strictly depends on the previous one. Star-shaped items are what P-SEARCH relies on for parallel decomposition: their ground-truth plans expose independent branches that the planning reward can score against parallel independence and that Node 3 subagents can run concurrently. Linear items are kept in the pool as negative examples for routing: their canonical strategy is direct (fast-track), and decomposing them is wasteful.

Table 7 contrasts a representative star item with a representative linear item drawn from the same synthesis run. We refer to the star example throughout

Parallel Question Synthesis Prompt

System Prompt

You are a master of multi-hop question design for training a Deep Research agent. Given a hub entity and several independent branches, synthesize ONE complex question that combines results from every branch.

Goal: produce a question that (i) requires every branch to be explored independently and in parallel, (ii) has a final answer obtained by combining or comparing branch results, (iii) describes entities through fuzzified attributes rather than naming them, and (iv) is unanswerable unless every branch is completed. Constraints: do not name the hub entity directly; each branch must correspond to one self-contained research sub-task; the final step must be one of {comparison, aggregation, constraint_join}.

User Prompt (Template)

Hub Entity: {hub_entity}

Branches: {branches_description}

Final Answer: {answer}

Difficulty: {difficulty} | Target Branch Count: {num_branches}

Output JSON: {

 "question": ..., "sub_questions": [...],

 "constraints": [...],

 "merge_type": "comparison|aggregation|constraint_join",

 "reasoning": "why this question requires parallel exploration"

}

Figure 2: **Parallel Question Synthesis prompt** (QSYNTH_{||}). Synthesizes one parallelizable multi-branch question per star-shaped subgraph; we show the compressed English template. Source code and full bilingual prompts will be released upon acceptance.

Canonical Parallel-Plan Generation Prompt

System Prompt

You generate a parallel execution plan (ground truth) for a complex multi-hop question. This plan supervises the Plan stage of an RL agent: the agent must learn to decompose questions into fully independent, truly parallel sub-tasks.

Critical Rule: every research sub-task must be independently executable; no sub-task may depend on another’s output. All research tasks start simultaneously with zero dependencies. The only allowed dependency is one final MERGE task that combines all research results. Instructions: break the question into N self-contained research sub-tasks (one per branch), each fitting into 1–3 search–read–reason cycles; add exactly one MERGE task; MERGE is the only task with a non-empty depends_on field.

User Prompt (Template)

Question: {question} | Answer: {answer}

Hop Chain: {chain_description}

Branches: {branches_description}

Sub-Questions: {sub_questions}

Output JSON: {

 "parallel_plans": [

 {"plan_id": "A", "task": "...", "depends_on": [], "expected_output": "..."},

 {"plan_id": "B", "...", "depends_on": []},

 {"plan_id": "C", "...", "depends_on": []},

 {"plan_id": "MERGE", "task": "combine A/B/C",

 "depends_on": ["A", "B", "C"], "expected_output": "final answer"

],

 "parallelizable_groups": [["A", "B", "C"]],

 "merge_strategy": "constraint_intersection|comparison|aggregation"

}

Figure 3: **Canonical Parallel-Plan Generation prompt** (PLANGEN_{||}). The output JSON is exactly the schema consumed by P-SEARCH-PLAN during decoupled-GRPO training.

the remaining appendices.

D.2 Sample schemas for P-SEARCH-REACT and P-SEARCH-PLAN

The two policies are trained from disjoint subsets of the same QA pool but with different sample

Field	Star example (used as running case)	Linear example (negative for routing)
structure_type	star	linear
difficulty	medium	medium
num_hops	10	6
num_parallel_branches	2	0
Hub / seed	A private university in Chicago’s Hyde Park, founded 1890 by an American industrialist	— (single chain)
Question (fuzzified, abridged)	A private university in Chicago’s Hyde Park (founded 1890) hosted a groundbreaking scientific achievement beneath its athletic field. How many years separate the year the site received National Historic Landmark status and the year a British novelist—who wrote a published work on the architecture of literary “plots”—was appointed to a British honour also referenced at the disposal ground near where the reactor was relocated?	A British WWII military section established clandestine pathways through occupied Europe; one route headed south to a landlocked neutral state whose separation from a centuries-old imperial entity was acknowledged in a mid-17th-century settlement of two parallel agreements; one ended a prolonged conflict between an Iberian power and a maritime trading confederation that was ultimately dissolved in what year?
Branch / hop sketch	A: Stag Field → CP-1 site → NHL designation 1965 . B: reactor relocation → Plot M → literary “plot” → E. M. Forster → Companion of Honour 1953 . MERGE: 1965 — 1953.	M19 → Switzerland → Peace of Westphalia → Treaty of Münster → Dutch Republic → Batavian Revolution.
Answer	12 (year difference)	1795 (year)
canonical_plan.strategy	decompose, is_parallelizable=true, parallel_groups=[[A, B]]	direct (fast-track), no decomposition needed
Role in training	Positive supervision for parallel decomposition; the MERGE dependency is the only sequential edge in the plan DAG.	Routing negative: rewards a fast-track answer; decomposing this question is penalized through C_{route} .

Table 7: **A star-shaped vs. a linear synthetic question.** Both items share the same metadata schema; what differs is the value of `structure_type` and the shape of `canonical_plan`. The star example contains two independently solvable branches whose results are aggregated by an explicit MERGE step, giving the Node 2 reward (Appendix F) a clear ground-truth notion of parallel independence. The linear example uses the same fields but with empty `parallel_plans`, so its planner reward favors a fast-track route.

schemas. P-SEARCH-REACT is trained directly on raw research questions, so each sample carries the full constraint and structure metadata needed by the process reward. P-SEARCH-PLAN is trained on *filtered* question-background pairs in which the upstream Node 1 background report is fixed (cf. Appendix C, Stage S6), so each sample additionally carries an auxiliary background string and a teacher-best plan label. Tables 8 and 9 list the fields, their roles, and their provenance.

The size gap (5,574 vs. 2,111) reflects the Stage S6 solvability filter: only questions for which at least one teacher rollout reached the correct answer (with a citation-level mock visit tool) are eligible for P-SEARCH-PLAN training, because the planner reward needs a non-trivial `teacher_best_plan` reference.

D.3 GRPO training hyperparameters

Both policies are optimized with decoupled GRPO under the Slime + Megatron + SGLang stack on $8 \times \text{H20}$ GPUs (4 training, 4 SGLang serving with $\text{TP}=2$). The final GRPO run for P-SEARCH-REACT took approximately 120 wall-clock hours, and the final GRPO run for P-SEARCH-PLAN took approximately 36 wall-clock hours on the same setup. Since both training and rollout-serving GPUs were reserved during these runs, this corresponds to roughly 960 and 288 allocated H20 GPU-hours, respectively. These estimates cover the final reported training runs and do not include earlier debugging or pilot runs. P-SEARCH-REACT uses

a longer maximum response length and a smaller per-prompt group than P-SEARCH-PLAN because its rollouts contain multi-turn tool-use trajectories, whereas P-SEARCH-PLAN emits a single JSON object. Table 10 summarizes the configuration used in our final runs.

E Node 1 ReAct GRPO Reward Details

This appendix expands the search-report reward used to train P-SEARCH-REACT, which is restated from §4.3 for convenience:

$$R_{\text{proc}}^{(1)}(\tau) = V(\tau) (0.5 R_{\text{exp}} + 0.5 R_{\text{util}}) R_{\text{eff}} R_{\text{dil}}. \quad (4)$$

We then describe the four-region confidence-aware correctness mapping that converts the process score into the final reward used by GRPO. All terms are bounded in $[0, 1]$.

E.1 Validity gate $V(\tau)$

$V(\tau)$ validates that the trajectory uses tools and produces a final-report block in the required format. It is a soft score in $[0, 1]$ aggregated from a tool-usage component and a final-format component, exactly as described in §4.3. As a consequence, a trajectory that issues no successful tool call (every `web_search` or `visit` action either fails the JSON schema check or returns an error) receives $V(\tau) = 0$, which zeros out the entire process reward in Eq. 4. Trajectories that exceed the per-rollout limits ($\text{MAX_TURNS}=20$, $\text{MAX_TOOL_CALLS}=10$) are also

Field	Source / how produced	Role in training
<i>Input messages (consumed by the policy at rollout time)</i>		
system	Fixed Node 1 role prompt	Defines the ReAct role: produce a structured background report under web-search and visit tools.
user	question_fuzzified from synthesis (Stage S5)	The actual research question shown to the agent; the unfuzzified question is held out and used only by the reward.
<i>Reward metadata (consumed by the reward function only)</i>		
qa_id	Hash of seed entity + chain id	Trajectory bookkeeping.
question	Original (unfuzzified) question (Stage S4)	Reference text for LLM-judge utility scoring.
answer	Hop-chain terminal fact (Stage S3)	Used by the four-region confidence-aware reward (§E).
constraints	Surface markers extracted in Stage S4	Drives the constraint-hit-rate factor in R_{exp} .
sub_questions	Per-hop natural-language sub-queries (Stage S4)	Diagnostic; not used by the reward but logged for analysis.
canonical_plan	Ground-truth plan (Stage S4)	Used by Node 2 training only; carried here for joint logging.
parallel_plans	Star-branch decomposition (Stage S4)	Used by Node 2 training only; carried here for joint logging.
hop_chain	Ordered list of {entity, url, fact} tuples (Stage S3)	Provides ground-truth evidence URLs for grounding checks.
difficulty / num_hops / structure_type / num_parallel_branches / num_constraints	Bucket assignment (Stage S2)	Used to balance training batches across difficulty buckets.

Table 8: **P-SEARCH-REACT training sample schema** (5,574 questions). The same JSONL row carries the input messages consumed by the policy and the reward metadata consumed by the reward function; the latter is never injected into the prompt.

Field	Source / how produced	Role in training
<i>Input messages</i>		
system	Node 2 role prompt pre-filled with {original_query, auxiliary_context}	Single message that already concatenates question and background; instructs the planner to emit a JSON MasterAnalysisResult.
<i>Label (used as reference, not for SFT)</i>		
label	Best teacher plan (teacher_best_plan)	Reference plan for the pointwise judge R_{pw} ; the policy is still trained by GRPO, not by token-level CE on this label.
<i>Reward metadata</i>		
qa_id	As above	Bookkeeping.
question / ground_truth / constraints	Inherited from the P-SEARCH-REACT pool	Used by the routing cap C_{route} to verify fast-track answers.
auxiliary_context	Node 1 background report from a teacher P-SEARCH run (Stage S6)	The <i>fixed</i> upstream input that makes Node 2 GRPO comparable across group samples.
difficulty / num_hops	Inherited from P-SEARCH-REACT pool	Used by the granularity-economy term R_{econ} (number of subtasks vs. optimal range).
canonical_plan	Ground-truth plan (Stage S4)	Used by the dependency cap C_{dep} and the structure-based pointwise judge.
teacher_best_composite	Composite reward of the best teacher rollout (Stage S6)	Confidence weight on the teacher signal; lower values down-weight the teacher hint inside C_{route} .
num_runs / correct_runs	Stage S6 rollout statistics	Used to derive the teacher-backoff agreement coefficient (§F).

Table 9: **P-SEARCH-PLAN training sample schema** (2,111 questions). The system message is the planner instruction with original_query and auxiliary_context already substituted in, so each rollout in a GRPO group sees an identical prompt and the comparison stays valid (§4.2).

Hyperparameter	P-SEARCH-REACT	P-SEARCH-PLAN
Backbone	Qwen3-8B (base)	Qwen3-8B (base)
Training set size	5,574	2,111
Epochs	2	2
Rollout batch size (prompts)	32	16
Group size G	8	16
Effective global batch (samples)	256	256
KL coefficient (low-var)	0.05	0.05
Clip range ϵ	0.2	0.2
Optimizer / schedule	Adam, constant	Adam, constant
Learning rate	5×10^{-7}	5×10^{-7}
(β_1, β_2) / weight decay	(0.9, 0.98) / 0.1	(0.9, 0.98) / 0.1
Sampling temperature	0.9	0.8
Max response length (tokens)	16,384	4,096
Max ReAct turns / tool calls	20 / 10	—
Hardware	8×H20 (4 train, 4 serve)	8×H20 (4 train, 4 serve)

Table 10: **GRPO training configuration.** Both runs share the global batch of 256 samples per step but allocate it differently: P-SEARCH-REACT uses fewer groups of longer trajectories, while P-SEARCH-PLAN uses more groups of shorter JSON outputs. KL is computed with the low-variance estimator; advantages are length-normalized within each group.

gated through V rather than through a separate term.

E.2 Exploration sub-score R_{exp}

$R_{\text{exp}} \in [0, 1]$ measures whether the trajectory casts a sufficiently broad net before committing to a final answer. It combines (i) a search-breadth factor on the number I of *unique* search intents (deduplicated query roots) and the number U of *unique* pages actually visited (deduplicated URLs), and (ii) a constraint-coverage factor based on the question metadata. The breadth factor is the geometric mean of two saturating sigmoids:

$$B = \sqrt{\sigma\left(\frac{I - \mu_I}{s_I}\right) \sigma\left(\frac{U - \mu_U}{s_U}\right)}, \quad (5)$$

with midpoints $(\mu_I, \mu_U) = (5, 3)$ and steepness $(s_I, s_U) = (0.80, 0.90)$, so that fewer than two unique searches or fewer than one visited page already saturate the term toward zero, while $I \geq 6$ unique intents and $U \geq 4$ unique pages saturate it toward one. The constraint-coverage factor is $0.5 + 0.5 \cdot \text{CHR}$, where $\text{CHR} \in [0, 1]$ is the constraint hit rate computed by string matching the trajectory’s evidence pool against the constraints field of the QA item. The full sub-score is $R_{\text{exp}} = B \cdot (0.5 + 0.5 \text{CHR})$, which simultaneously rewards diversified search and explicit constraint targeting.

E.3 Utility sub-score R_{util}

$R_{\text{util}} \in [0, 1]$ measures the usefulness of the evidence-grounded final report for the downstream Node 2 planner and Node 3 subagents.

It is the LLM-judge score from a frozen reward model that returns two scalars on the trajectory: `state_targeting` (how well the searches targeted the question’s constraints) and `grounded_report_utility` (how usable the final report is by a downstream agent). We aggregate them as

$$R_{\text{util}} = 0.55 s_{\text{state}} + 0.45 s_{\text{ground}}. \quad (6)$$

The exact judge prompt is shown in Figure 4. If the judge fails to return parseable JSON, R_{util} defaults to 0 and is logged as a judge failure.

E.4 Efficiency factor R_{eff} and diligence factor R_{dil}

The two remaining multiplicative factors penalize redundant or low-yield search and premature termination, respectively. Let n_{search} and n_{visit} be the total number of search and visit calls (including duplicates and failures), and recall I, U from §E.2. We define

$$R_{\text{eff}} = \min\left(\frac{I}{\max(n_{\text{search}}, 1)}, \frac{U}{\max(n_{\text{visit}}, 1)}\right), \quad (7)$$

which equals one when every issued query and every visited page contributed a new intent or URL, and decreases as the agent re-issues near-duplicate queries or revisits the same pages. The diligence factor R_{dil} caps trajectories that terminate too early: a trajectory with fewer than seven total tool calls is penalized to $R_{\text{dil}} = 0.30$, while $R_{\text{dil}} = 1$ otherwise.

E.5 Final reward and confidence-aware correctness regions

We map $R_{\text{proc}}^{(1)}$ to the GRPO reward through a four-region partition determined by (i) whether the final answer is judged correct against the ground truth, and (ii) the agent’s self-reported confidence $c \in [0, 1]$ extracted from the final report. Two thresholds are used: a high-confidence threshold $c_{\text{hi}} = 0.65$ and an over-confidence threshold $c_{\text{ovc}} = 0.90$. The four regions are summarized in Table 11.

The reward judge that produces s_{state} and s_{ground} , as well as the final-answer correctness label used in Table 11, share the same rubric template summarized in Figure 4.

Node 1 reward judge prompt (summarized)

System Prompt

You are a Search Quality Auditor evaluating a ReAct research agent. Score two dimensions on the agent’s trajectory using the rubric below; output JSON only.

Inputs

Role: {role_description}

Question: {question}

Confidential reference: {constraints_formatted} (total N), {structure_type}, {num_hops}.

Agent performance: {search_queries}, {observation_summary}, {final_report}.

Rubric

(i) state_targeting (weight 0.55): how precisely the searches target the listed constraints; reward queries that increase the constraint hit rate, penalize generic restatements of the question.

(ii) grounded_report_utility (weight 0.45): how useful the final report is for a downstream planner; reward citation grounding (each major claim backed by a visited URL), penalize unsupported assertions and verbatim copying.

Output JSON

{ state_targeting: {score, constraints_hit, reasoning};

grounded_report_utility: {score, grounding_ratio, reasoning} }

Figure 4: **Node 1 reward judge prompt (summary)**. Both dimensions are scored in $[0, 1]$ on the trajectory; the aggregate $R_{\text{util}} = 0.55 s_{\text{state}} + 0.45 s_{\text{ground}}$ enters the process reward in Eq. 4. The reference fields (constraints, structure_type, num_hops) come from the QA metadata schema in Appendix D.2 and are visible only to the judge, never to the policy.

Region	Condition	Reward
A (sufficiently grounded correct)	correct, $c \geq c_{\text{hi}}$	$\max(R_{\text{proc}}^{(1)}, 0.75)$, then capped at 1.0 and bonused by +0.15
C_d (under-evidenced correct)	correct, $c < c_{\text{hi}}$	$\max(R_{\text{proc}}^{(1)}, 0.50)$, capped at 0.75
B (overconfident wrong)	wrong, $c \geq c_{\text{ovc}}$	$R_{\text{proc}}^{(1)} \cdot (1 - c)$
C (uncertain)	wrong, $c < c_{\text{ovc}}$	$R_{\text{proc}}^{(1)}$

Table 11: **Confidence-aware correctness mapping**. Sufficiently grounded correct answers receive the highest range; correct but under-evidenced answers are capped lower so that the model learns to keep exploring; overconfident wrong answers are discounted by their own confidence; uncertain wrong answers retain the unmodified process reward and are still trained on for behavior shaping.

F Node 2 Plan GRPO Reward Details and JSON Schema

This appendix expands the planning reward used to train P-SEARCH-PLAN, restated from §4.3:

$$R_{\text{FT}}^{(2)} = C_{\text{route}}, \quad (8)$$

$$R_{\text{DEC}}^{(2)} = C_{\text{route}} C_{\text{dep}} (0.85 R_{\text{pw}} + 0.15 R_{\text{econ}}). \quad (9)$$

The branch is selected by the planner’s own routing flag `is_fast_track` (a Boolean field in the JSON output, §F.6). Plans that fail JSON schema validation receive zero reward and are excluded from advantage normalization within their GRPO

group. The same reward is used for Node 2 initial planning and Node 4 phase-1 Judge-style verification, with only the inputs differing (raw question vs. verification state).

F.1 Routing cap C_{route}

$C_{\text{route}} \in [0, 1]$ encodes whether the chosen route (fast-track vs. decompose) is appropriate. We compute it by comparing the planner’s route flag and, on the fast-track branch, its `fast_track_answer` against the ground-truth answer; on the decompose branch we additionally use a teacher-backoff signal computed from Stage S6 statistics (Appendix C). The caps are summarized in Table 12.

Route	Outcome	C_{route}
Fast-track	<code>fast_track_answer</code> matches ground truth	1.00
Fast-track	<code>fast_track_answer</code> present but wrong	0.15
Fast-track	<code>fast_track_answer</code> empty / missing	0.10
Decompose	teacher rollouts on the same item also chose decompose	1.00
Decompose	item was solvable by fast-track but planner decomposed	0.90

Table 12: **Routing caps**. The cap penalizes overconfident fast-track guesses and overly conservative decomposition; it is the only term that depends on the ground-truth answer at training time.

F.2 Dependency cap C_{dep}

C_{dep} gates plans that contain obvious sequential dependencies between subtasks, since such plans cannot be executed in parallel by Node 3. We use

a fixed list of nine regular-expression patterns over the concatenated subtask texts (e.g. phrases such as “based on the result of”, “after task X”, “using the year found in”). If any pattern hits, $C_{\text{dep}} = 0.6$; otherwise $C_{\text{dep}} = 1.0$. The same dependency check is also used to compute the routing-cap row “solvable by fast-track but decomposed” on linear items: a linear item that gets decomposed almost always triggers a dependency match.

F.3 Pointwise plan-quality judge R_{pw}

$R_{\text{pw}} \in [0, 1]$ is an LLM-judge score on two dimensions of the planner’s JSON output (Figure 5):

$$R_{\text{pw}} = 0.55 s_{\text{plan}} + 0.45 s_{\text{indep}}, \quad (10)$$

where plan_quality (s_{plan}) audits branch coverage, actionability, constraint identification, and answerability; $\text{parallel_independence}$ (s_{indep}) is the central objective and audits whether the listed subtasks have a non-trivial DAG dependency. The judge sees the original question, the auxiliary background context, the planner’s full JSON output, and the canonical reference plan from the QA metadata; it never sees the ground-truth final answer.

F.4 Granularity-economy term R_{econ}

$R_{\text{econ}} \in [0, 1]$ discourages excessive or repetitive subtasks. For each difficulty bucket we define an optimal range of subtask count $[k_{\text{min}}, k_{\text{max}}]$:

$$\begin{aligned} \text{easy} &= [2, 4], & \text{medium} &= [3, 6], \\ \text{hard} &= [4, 7], & \text{extreme} &= [5, 9]. \end{aligned}$$

Let k be the number of leaf subtasks in the planner’s output. $R_{\text{econ}} = 1$ if $k \in [k_{\text{min}}, k_{\text{max}}]$, and decays linearly to 0 at $k = 0$ or $k \geq 2k_{\text{max}}$. The hard cap on subtask count is $\text{PROMPT_MAX_SUBTASKS} = 9$; outputs above this cap are treated as JSON-invalid.

F.5 Plan judge prompt

Figure 5 summarizes the prompt template used by the plan judge. The full prompt body (rubric examples and edge-case clarifications) is omitted for space; the figure itself is placed alongside the Node 2 reward description for side-by-side reference.

F.6 MasterAnalysisResult JSON schema

The planner is constrained to emit a single JSON object of type `MasterAnalysisResult`. The same schema is consumed by Node 2 (initial planning) and Node 4 (phase-1 verification planning); the

only field that distinguishes the two phases is the value of `is_fast_track` and the contents of subtasks. Table 13 lists the fields and their roles in the reward.

Field	Type	Description
<code>is_fast_track</code>	bool	Routes to Eq. 8 when true, else Eq. 9.
<code>fast_track_answer</code>	str	Final answer on the fast-track branch; empty otherwise.
<code>subtasks</code>	list	Each entry tagged [Type A/B/C]; its size and dependency pattern drive C_{dep} and R_{econ} .
<i>Sub-fields under analysis (consumed by the pointwise judge R_{pw})</i>		
<code>query_type</code>	enum	factual / computational / aggregational / verification.
<code>all_constraints</code>	list	Surface constraints extracted from the question.
<code>primary_hypothesis</code>	str	Most likely answer given the auxiliary context.
<code>alternative_hypotheses</code>	list	Falsifiable alternatives the subtasks must rule out.
<code>key_facts_from_auxiliary</code>	list	Facts already confirmed in the auxiliary context.
<code>answer_form</code>	str	Expected form of the final answer.
<code>verification_strategy</code>	str	How subtask outputs combine into the final answer.

Table 13: **MasterAnalysisResult** schema. The reward in Eq. 9 reads `is_fast_track` and `fast_track_answer` for C_{route} , the subtasks block for C_{dep} and R_{econ} , and the full analysis block for R_{pw} .

G Evaluation Judge Prompt

For benchmarks that do not provide an official exact-match metric (BrowseComp, BrowseComp-ZH, GAIA, Xbench-DeepSearch, HLE), we run the same LLM-as-Judge that is used to extract the final answer from the agent’s report and to compare it against the gold answer in a single call. For all internally evaluated P-SEARCH configurations and ablations reported in Tables 2–4, the judge model is fixed to `glm-5`. We use a single judge model across benchmarks rather than selecting benchmark-specific judges. The full rubric is summarized in Figure 6.

Inputs and outputs. The judge receives three strings: the question, the agent’s full `final_report`, and the gold answer. It returns a JSON object with fields `extracted_final_answer`, `reasoning`, and `correct`; if JSON parsing fails three times in a row, we fall back to a string match on “correct”: “yes” and log the case as a judge failure. The

Node 2 plan judge prompt (summarized)

System Prompt

You are an expert evaluator of research-plan quality. Score two dimensions on the planner’s structured output using the rubric below; output JSON only.

Inputs

Original question: {original_query}
Auxiliary context: {auxiliary_context}
Planner output: {plan_json}
Reference plan: {canonical_plan}
Metadata: {difficulty, num_hops, num_parallel_branches}.

Rubric

(i) plan_quality (weight 0.55): branch coverage of the question’s constraints, subtask actionability (clear search target and expected output), explicit constraint identification, and end-to-end answerability if every subtask succeeds.
(ii) parallel_independence (weight 0.45, the central objective): each subtask must be independently solvable; reward true sibling parallelism, penalize hidden DAG edges (later subtasks reusing entities discovered by earlier ones).

Output JSON

```
{ plan_quality: {score, reasoning};  
  parallel_independence: {score, reasoning};  
  overall_judgment: {...} }
```

Figure 5: **Node 2 plan judge prompt (summary)**. The same two-dimensional rubric is reused at inference time inside Node 4’s phase-1 Judge step, with the verification-state JSON replacing the planner output. The judge never sees the ground-truth final answer; routing correctness is handled separately by C_{route} in Eq. 8–9.

same call therefore performs (i) free-text answer extraction (replacing fragile regex extractors), and (ii) correctness adjudication; this matches the official LLM-as-Judge protocol used by HLE and is consistent with public BrowseComp evaluation scripts.

Why surface-form errors do not propagate. A common worry with LLM judges is that they are too strict on surface form (case, hyphen, transliteration) or too loose on entity identity. Our prompt closes both holes by reducing the rubric to two short, mutually exclusive lists with a fixed decision priority. The *accept* list contains four categories and each is anchored on a *verifiable equivalence*: same real-world referent under different surface form, same time point in a different written format, same entity together with one or more correct co-attributes, and droppable disambiguation tags in the gold answer. The *reject* list contains four categories that block any concrete numeric disagreement, identity disagreement, ordering disagreement, and non-answers (refusal, range-only answer, off-topic answer). The decision priority forces the judge to scan the accept list first and lock the verdict if any category matches, with the reasoning required to cite both the matched category and the equivalence evidence; an accept verdict can therefore not be overturned by generic objections

(“the names are not literally identical”, “format differs”) but is also not allowed to be invented when no concrete equivalence holds.

Robustness in practice. The rubric is designed to make numeric and entity-level disagreements easy to reject while reducing surface-form false negatives: every accept category requires a concrete surface-form or co-attribute equivalence and every reject category is grounded in a quantifiable difference. In our manual audit of judged outputs, we did not observe cases in which the judge accepted an answer with a different year/date/numeric value or a swapped entity, which is consistent with the design above. We additionally re-judge each evaluation run with the same prompt to verify determinism: re-judgement changes the verdict on $< 1\%$ of items and these flips are restricted to genuinely borderline equivalence cases (e.g. “early one round” vs. “one round in advance”), never to clear-cut numeric or entity differences.

Cross-judge agreement. Since the judge is fixed to glm-5 and one of the evaluated configurations also uses GLM-5 as its backbone, we additionally check whether the reported scores depend on the judge family. We re-scored the same 289 BrowseComp-ZH runs with two non-GLM judges, qwen3-max and kimi-k2.6, reusing the identical judge prompt and parsing logic and calling each

judge through an independent API key. The resulting accuracies are 75.1% (GLM-5), 75.4% (Qwen3-Max) and 74.7% (Kimi-K2.6), and the three judges return exactly the same verdict on 286 of 289 items (98.6% agreement); the three disagreements are borderline equivalence cases of the kind discussed above. The spread across judge families is thus below one point and does not follow the backbone identity, which is inconsistent with self-preference inflation. As with the seed-level check in §5.1, we treat this as a stability check on the evaluation protocol rather than as a variance or significance estimate.

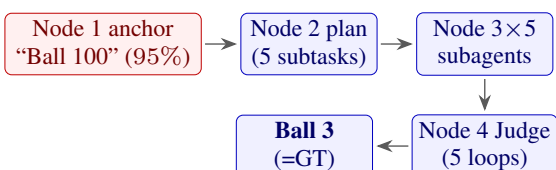
H Detailed Case-Study Logs

This appendix presents three anonymized case studies drawn from the GAIA and BrowseComp evaluation runs reported in Tables 2–4 (two from GAIA, one from BrowseComp). All cases share one color convention: **red** boxes mark a path that was *not yet verifiable* (Node 1’s anchor, or a same-backbone self-plan baseline), **blue** boxes mark the Node 2 plan, Node 3 subagent, or Node 4 verification outputs that closed the gap, and gray boxes mark the question, ground truth, and raw retrieval. The cases instantiate, in order, a buggy local simulation corrected through five judge loops (§H.1), Judge-loop closure on under-determined evidence (§H.2), and a multi-constraint falsification grid in which a dedicated Planner overturns the self-plan baseline via targeted [Type C] subtasks (§H.3). All quotes are condensed from the actual trace logs (timestamps, retry chatter, and tool-result boilerplate removed); the conclusions depend only on the visible evidence.

H.1 Numerical anchor recovery: a buggy local simulation overturned by parallel re-implementations

Question `gaia_ec09fa32` | Ground truth: 3

Pick That Ping-Pong (abridged). 100 numbered balls 1...100 feed in order onto a 3-slot platform. At each step a uniformly random piston fires: piston-1 ejects slot-1 (you win iff your pick is ejected); piston-2 ejects slot-2 and *releases* slot-1; piston-3 ejects slot-3 and *releases* slot-1. Released balls do not count as a win. Which ball maximizes the ejection probability?



Node 1: high-confidence simulator trap

P-SEARCH-REACT, 3589 s, 40 tool calls. The ReAct loop writes a Python simulator, runs 20,000 trials and reports “Ball 100 ejected 100% of the time”; web search returns no external solution. **Final answer: 100, confidence 95%** — justified by trial consistency alone, with no check of the simulator’s edge cases once the ramp empties and the platform holds fewer than three slots.

Node 2 (P-SEARCH-PLAN, 7.2s): falsifiable plan, is_fast_track=False

primary_hypothesis: Ball 100;
 alternative_hypotheses: {Ball 99, 1, 3};
 verification_strategy: “re-derive the ejection distribution from the mechanism rather than from the auxiliary simulator.” Five subtasks: [Type C] × 3 verify the 100% claim for Ball 100, test the lower-rate claims for Balls 99/1/3, and seek an alternative explanation; [Type E] enumerate the positions the chosen ball can occupy when the ramp is empty; [Type F] cross-check the mechanism externally.

Node 3 / Node 4 closure

Subagents 2, 4, and 5 independently re-implement the simulator and converge on $P(\text{eject Ball 3}) \approx 62.7\%–92.7\%$ versus $P(\text{eject Ball 100}) \approx 55.4\%–60.2\%$; subagents 4 and 5 also surface an external blog (PromptQL) that names Ball 3 and diagnoses Ball 100 as a common “intuitive trap”. Node 4 marks `evidence_sufficient = False` while every back-reference still points to the same buggy auxiliary, then re-plans until each constraint has subagent-quotable support.

Node 5 final report

“The convergence of multiple independent simulations and an external blog provides strong evidence for Ball 3; the 100% claim for Ball 100 is incorrect because Ball 100 can be *released* when piston-2 or piston-3 fires while it sits in slot-1.” Final answer: 3 (confidence 85%).

Takeaway. Single-branch ReAct can sustain a 95%-confidence wrong answer when its only evidence is its own buggy computation: large- N self-consistency amplifies the error instead of checking it. The framework breaks the loop twice — Node 2 promotes “simulator correctness” to an explicit [Type C] subtask, and Node 4 refuses closure while every constraint cites the same auxiliary — so the answer flips only after independent re-implementations and an external source agree (§3.3, §3.5).

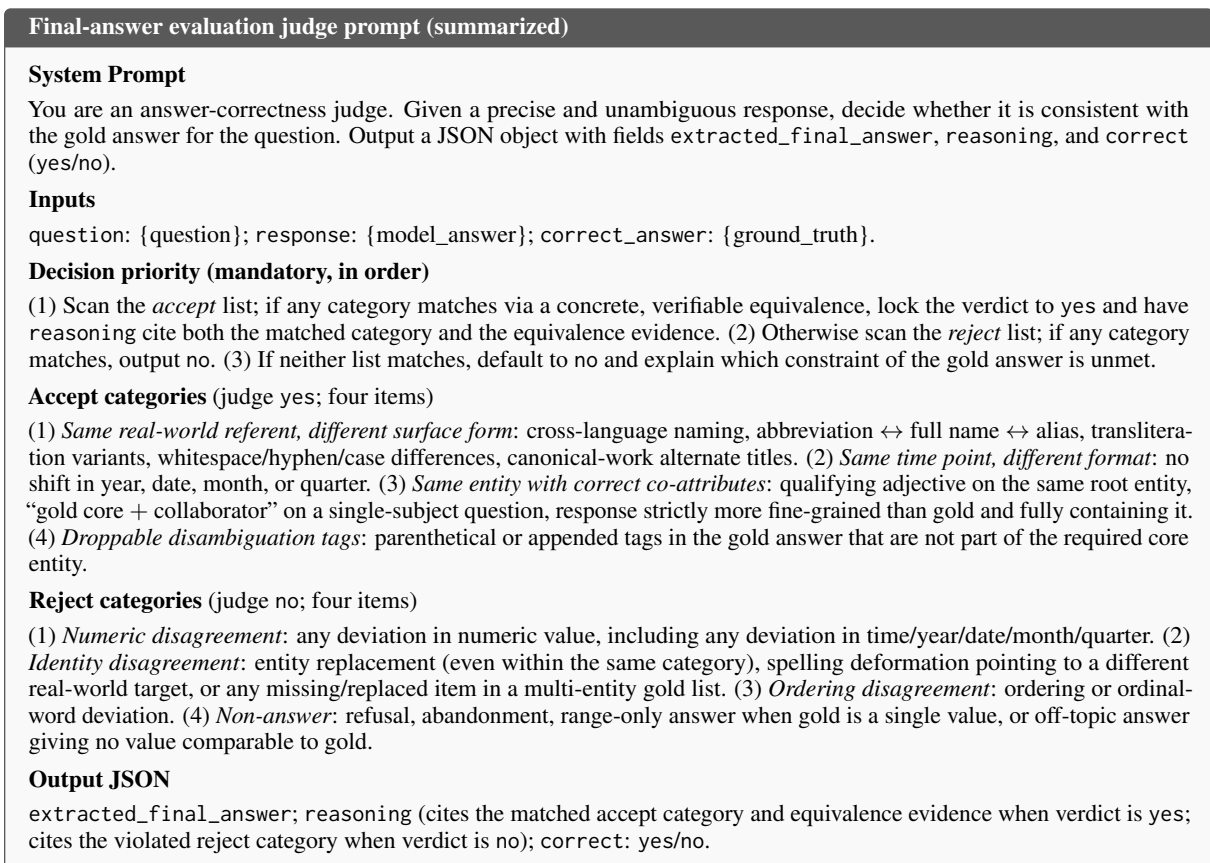
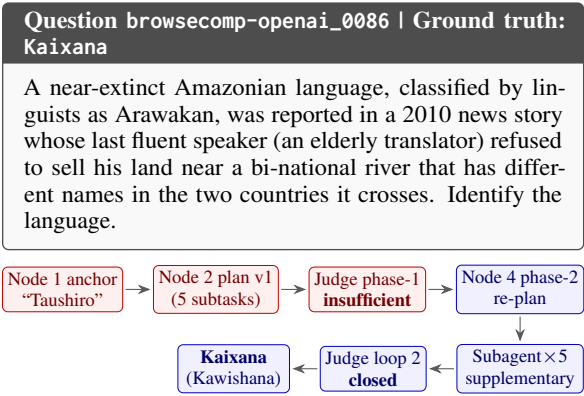


Figure 6: **Final-answer evaluation judge prompt (summary)**. A single call produces both the extracted answer and the correctness verdict. The two-list rubric reduces the original prompt to four accept categories and four reject categories, each anchored on a verifiable equivalence or a quantifiable difference; the accept-first decision priority eliminates surface-form false negatives without admitting numeric or entity-level false positives. The judge is held fixed to glm-5 for all internally evaluated P-SEARCH configurations and ablations.

H.2 Judge-loop closure: under-determined evidence triggers a second pass



Node 1 / Node 2 first pass: high-confidence wrong anchor

primary_hypothesis: “Taushiro (a.k.a. Pinche / Pinchi) — Peruvian Amazon language isolate near the Tigre River, last speaker Amadeo García García.” This anchor satisfies the “last-speaker translator” and “2010 news story” constraints but never tests the *bi-national river with two names* constraint: the Tigre River keeps the same name across Ecuador and Peru.

Node 4 phase-1 Judge: re-opens the river constraint

verification_state: *constraint [river has different names in two countries] = unverified*. The Judge does not have to recompute the answer; it only needs to detect that the chain “Tigre = Tigre across Ecuador / Peru” fails the constraint. The router emits a fresh MasterAnalysisResult with five new Type-C subtasks targeting candidate Amazonian languages whose host river is renamed across the border.

Node 3 supplementary subagents (loop 2): falsify and rebuild

Subagent 2: “Japurá River ... known as Caquetá in Colombia” (Wikipedia) ⇒ Japurá / Caquetá satisfies the bi-national-river constraint.
Subagent 5: rules out Ayapaneco (Mexico, no international river).
Cross-reference: “Kaixana classified as Arawakan” (linguistic database); “Raimundo Avelino, 78 in 2008, Japurá municipality, refused to sell his land” (Christian Science Monitor 2010); “Kaixana Translator Refuses Land Sale” (archive).

Node 4 phase-2 + Node 5 final report

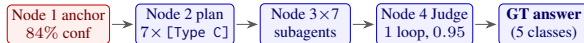
All five constraints (river renaming, < 1000 speakers, 2010 report, translator-refuses-sale, Arawakan classification) are now backed by independent quotes; verification_state.is_complete = true. Final answer: **Kaixana (Kawishana)**.

Takeaway. The first plan was *coherent but under-determined*: the early anchor satisfied four of five constraints, and a single-pass system would have stopped here. The Judge’s role (§3.5) is precisely to refuse closure when even one surface constraint has not been independently checked; the second pass is cheap because Node 4’s re-plan reuses the already-verified constraints and only allocates subagents to the unresolved one.

H.3 Multi-candidate falsification grid: a dedicated Planner flips a same-backbone self-plan error

Question gaia_9e1fc53b | **Ground truth:** Death Knight, Hunter, Paladin, Priest, Warlock

A 5-man WotLK Classic group (1 tank, 1 healer, 3 DPS), with two *plate* and two *cloth* wearers, downs a final boss. Both tank and healer cast *holy* spells; one DPS uses *ice*, another uses *fire*; a *bear* from the group attacks the boss; *Metamorphosis* is cast; the *Kilt of the Forgotten One* drops but *no one* can use it. All classes are unique. Name the five classes in alphabetical order.



self-plan ablation (Node 2 swapped to the same Tongyi-DR-30B backbone): plan-level mis-anchoring

With Node 1 evidence held identical, swapping Node 2 to the same 30B backbone mis-aligns three constraints at planning time: (i) reads “bear” as Druid bear-form rather than a Hunter pet; (ii) keeps Demon Hunter alive for “Metamorphosis” even though Demon Hunters post-date WotLK Classic; (iii) flags the kilt contradiction (“no one can use it”) but never converts it into a falsification subtask. The resulting `primary_hypothesis` = “Death Knight, Demon Hunter, Druid, Hunter, Priest”; only 4 broad-keyword subtasks are issued (1 Type-A, 1 Type-B, 1 Type-C, 1 Type-F). Node 5 then drifts further to “**Druid, Mage, Paladin, Priest, Warlock**” — two entities replaced (Druid for Hunter, Mage for Death Knight) — and is judged no (Table 3, Tongyi-DR-30B *self-plan* row).

Node 2 with the dedicated P-SEARCH-PLAN (29.2 s, is_fast_track=False): targeted falsification grid

`primary_hypothesis`: “Death Knight, Hunter, Paladin, Priest, Warlock”; `verification_strategy`: “verify each class meets the armor, spell-school, role and loot constraints, and that the leather kilt is unusable because the group contains no leather wearer.” Seven [Type C] subtasks, each a single-claim falsification rather than a broad search: (1) Metamorphosis = Warlock-only in WotLK Classic; (2) bear = Hunter pet (not Druid form); (3) Kilt of the Forgotten One = leather and unusable for the five candidates; (4) DK Frost (= ice) ∧ Warlock Destruction (= fire); (5) Holy school = Paladin ∪ Priest only; (6) Hunter is the unique bear-pet class; (7) armor uniqueness check (2 plate + 2 cloth + 1 mail).

Node 3 (×7 subagents): candidate–constraint falsification matrix

Class	armor (2P/2C/1M)	holy/ice/fire	bear / Metamorphosis	kilt unusable
<i>retained by [Type C]</i>				
Death Knight	plate ✓	ice (Frost) ✓	—	not leather ✓
Hunter	mail ✓	—	bear pet ✓	not leather ✓
Paladin	plate ✓	holy (tank) ✓	—	not leather ✓
Priest	cloth ✓	holy (heal) ✓	—	not leather ✓
Warlock	cloth ✓	fire (Destruction) ✓	Metamorphosis ✓	not leather ✓
<i>rejected by [Type C]</i>				
Druid	leather ×	no holy ×	bear is form, not pet ×	could equip kilt ×
Demon Hunter	leather ×	—	post-WotLK class ×	could equip kilt ×
Mage	cloth	ice or fire	—	breaks 2-cloth count ×

Node 4 + Node 5 final report

Judge phase-1 marks all 7 workers
thoroughness=thorough, sources=high;
evidence_sufficient=True, confidence=0.95;
phase-2 skipped. Final answer: **Death Knight, Hunter, Paladin, Priest, Warlock** (confidence 95%).

Takeaway. With Node 1 held identical (the same 30B ReAct backbone, 65 tool calls, 84% confidence) the only variable separating the failed and successful runs is the Planner. The same-backbone *self-plan* retains two impossible candidates (Druid for “bear”, Demon Hunter for “Metamorphosis”), issues only four broad-keyword subtasks, and lets Node 5 hallucinate a third wrong class (Mage); the dedicated P-SEARCH-PLAN turns the same evidence into a 7-cell falsification grid that disqualifies every alternative without needing a Judge re-plan. This is the structural source of the planner-transfer gain in Table 3: a sharper Plan, not a deeper Judge, is what flips the answer.