DecomposeR: Planner-Centric Reinforcement Learning for Deep Research with Structure-Aware Reward

Anonymous authors
Paper under double-blind review

Abstract

Deep research tasks require LLMs to plan what to investigate, retrieve evidence, and synthesize long-form answers across multiple branches of inquiry. Existing training paradigms either rely on short-form verifiable QA as a proxy or optimize monolithic long trajectories, which makes planning and execution difficult to disentangle and yields weak credit assignment for the planning process. We propose DecomposeR, a planner-centric deep research framework that represents research plans as typed directed acyclic graphs (DAGs), allowing planning to be made explicit, structured, and rewardable. We train a Qwen3-8B model in two stages: planner reinforcement learning (RL) first learns graph structure and query decomposition to improve research planning, and answerer reinforcement learning (RL) then learns branch-level execution and final synthesis conditioned on the learned plan. By assigning rewards to explicit planner tokens and structured components rather than to a flat trajectory, DecomposeR enables finer-grained optimization of planning while reducing the ambiguity of end-to-end training. Experiments show that DecomposeR-8B outperforms the strongest comparable open baselines of similar size by 5.1–8.0 points across all three benchmarks (DeepResearchBench, HealthBench, ResearchQA-Mini). It also rivals open models roughly 4× larger in size (51.7 average, vs. 51.4 and 51.1 for the two strongest 30–32B open systems), due to improved planning and answering.

1 Introduction

Deep research (DR) requires planning what to investigate, issuing multiple searches, selecting evidence, and synthesizing findings into a citation-grounded long-form report. Such workflows are increasingly important for scientific literature synthesis, clinical evidence aggregation, and general investigative tasks (Asai et al., 2024a; Singh et al., 2025; Arora et al., 2025; Du et al., 2025). Frontier proprietary systems demonstrate the practical value of this workflow (OpenAI, 2025; Comanici et al., 2025; Perplexity Team, 2025), while open alternatives either wrap capable LLMs in training-free retrieval pipelines (Li et al., 2026; 2025b) or post-train smaller models with reinforcement learning (Jin et al., 2025; Song et al., 2025; Chen et al., 2026; Mei et al., 2025).

Despite this progress, current RL-trained DR systems share two reward-side limitations that bound their training signal. First, credit assignment is ambiguous. Reasoning, search, evidence selection, branch synthesis, and final writing are interleaved in a flat ReAct-style trajectory (Yao et al., 2023), and a single scalar reward derived from the final answer is propagated across every model action. A weak final answer therefore cannot be attributed to a specific failure point so the gradient cannot direct improvement towards the responsible component. Second, reward is sparse. Intermediate planning, search, and synthesis decisions receive little to no direct supervision (Jin et al., 2025; Song et al., 2025; Chen et al., 2026); every intermediate action is shaped only through its eventual effect on the terminal answer, which both slows learning and destabilizes credit propagation across long DR trajectories.

We introduce DecomposeR, which addresses both limitations through a single design principle: structure-aware reward modeling for deep research. The research plan is materialized as an explicit typed DAG (Figure 1) in which search nodes issue web queries, aggregate nodes synthesize branch-level conclusions, and a

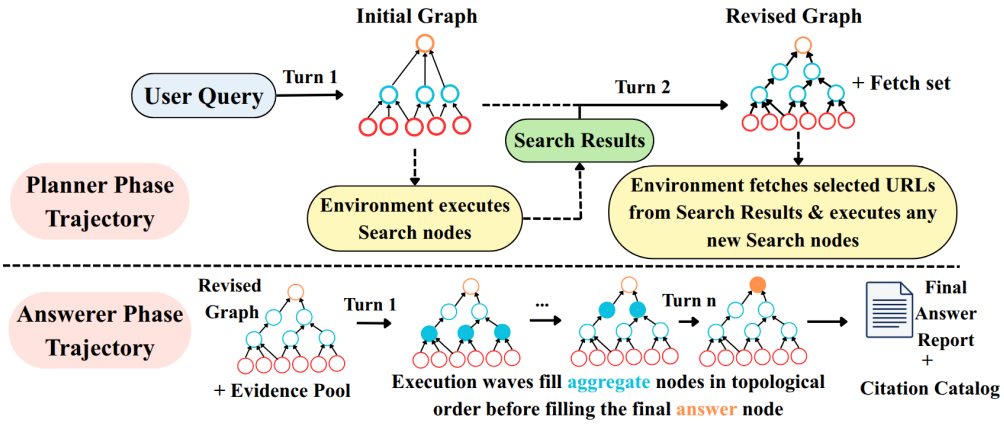

Figure 1: DecomposeR rollout structure. The planner emits an initial typed DAG, receives search results, revises the graph and fetch set, and then the answerer executes the revised graph in topological waves before writing the final answer.

terminal answer node integrates branches into the final report. Because every plan component is individually addressable, reward can be assigned to the corresponding part of the research process rather than only to the final report. Planner rewards score rubric coverage over plan-node fields, search quality over search nodes, and structural behavior such as branch breadth and evidence reuse; answerer rewards score branch-level execution and final synthesis. This design improves credit assignment by tying reward to the component responsible for each decision, and reduces sparsity by replacing a single trajectory-end scalar with signals distributed across the decomposition.

We evaluate DecomposeR on three long-form benchmarks: DeepResearchBench, ResearchQA-Mini, and HealthBench. DecomposeR improves over strong comparable open baselines by 5.1–8.0 points despite a substantially smaller end-to-end training budget.

Our contributions are: (1) **Structure-aware reward modeling for deep research**: we materialize the DR plan as a typed DAG whose structural components are directly addressable by reward functions, allowing separate signals to optimize the corresponding plan and execution components rather than being read off a trajectory terminus; (2) **DecomposeR framework**: we instantiate these rewards with typed decomposition graphs and staged reinforcement learning, making search breadth, evidence reuse, branch integration, execution fidelity, and final answer quality addressable optimization targets; and (3) **Empirical validation**: we evaluate on three long-form benchmarks, with analysis showing that the reward components, structured decomposition interface, and search-conditioned revision each contribute to the final gains.

2 Related Work

Deep research agents. Recent DR systems use web-scale retrieval and long-form synthesis to answer open-ended queries. Frontier systems such as OpenAI Deep Research rely on strong base models and inference-time orchestration (OpenAI, 2025). WebWeaver moves toward more explicit structure through dynamic outlines, evidence memory, and citation-grounded writing, but remains an inference-time dual-agent framework rather than an RL objective over a typed plan (Li et al., 2025b). WebThinker interleaves thinking, searching, navigation, and drafting inside a web-enabled reasoning loop and improves tool use with online preference optimization (Li et al., 2026). WebSailor and WebExplorer focus on long-horizon web agents, constructing high-uncertainty or evolved web-navigation tasks and training agents to handle many tool calls (Li et al., 2025a; Liu et al., 2025). These systems demonstrate the importance of search and synthesis, while DecomposeR focuses on making the research plan itself a structured, separately rewarded policy object.

RL for search and tool use. GRPO and RLVR-style post-training have been applied to reasoning, search, and tool use (Shao et al., 2024; Guo et al., 2025). Search-R1 trains models to interleave reasoning and search with retrieved-token masking and outcome rewards (Jin et al., 2025), while R1-Searcher studies outcome-based

RL for invoking search without process supervision (Song et al., 2025). ReSearch similarly treats search calls as part of the reasoning chain using explicit <search> and <result> tags, and masks retrieved results from the policy-gradient loss (Chen et al., 2026). O^2-Searcher targets both open-ended and closed-ended QA through a local search environment and task-aware rewards (Mei et al., 2025). ParallelSearch addresses an orthogonal efficiency problem by rewarding models for decomposing independent sub-queries that can be searched in parallel (Zhao et al., 2025). DecomposeR differs by exposing the full research plan as a typed graph and assigning reward to graph components before answer synthesis is trained, rather than treating the search trajectory as a flat sequence of tool actions.

Query decomposition and retrieval-augmented generation. Prior decomposition methods decompose complex questions into subproblems through prompting or modular inference (Khot et al., 2023; Wang et al., 2023; Trivedi et al., 2023). Retrieval-augmented generation systems learn when and what to retrieve (Asai et al., 2024b; Jiang et al., 2023). DecomposeR brings these ideas into RL for open-ended DR by making decomposition a trainable graph policy rather than an unobserved latent reasoning trace.

3 Methodology

3.1 Problem Formulation

Existing RL-trained DR systems model the agent rollout as a flat sequence

$$\tau_{\text{flat}} = (x, m_1, o_1, \ldots, m_T, o_T, y), \tag{1}$$

in which model actions m_t and tool observations o_t interleave until a final answer y is emitted, and a single scalar reward $R(x, y)$ is propagated through every m_t for policy update. The limitations identified in Sec. 1 are directly readable from this formulation. First, every m_t – whether it carries a planning decision, a search query, or a synthesis sentence – is updated under the same R, so the gradient cannot localize the failure to a specific role. Second, $R(x, y)$ supervises the entire $\{m_t\}_{t=1}^{T}$, leaving each intermediate m_t shaped only through its eventual effect on the terminal y. Third, m_t is an unstructured token sequence, so dependencies between subtasks and the reuse of evidence across branches have no native representation.

DecomposeR replaces this flat trajectory with a factored one in which an explicit plan $\mathcal{G}$ mediates between a planner policy π_P and an answerer policy π_A:

$$\tau = (x, \mathcal{G}_0, Z, \mathcal{G}_1, F, O_{1:K}, y), \tag{2}$$

where $\mathcal{G}_0$ is an initial plan emitted by π_P, $Z = \mathcal{E}(\mathcal{G}_0)$ are search observations returned by the environment $\mathcal{E}$, $\mathcal{G}_1$ is the planner's revised plan, F are fetched documents, $O_{1:K}$ are intermediate outputs produced by π_A in K topological waves, and y is the final answer. The joint trajectory distribution factorises as

$$\pi(\tau|x) = \pi_P(\mathcal{G}_0|x)\, \mathcal{E}(Z|\mathcal{G}_0) \cdot \pi_P(\mathcal{G}_1, F|x, \mathcal{G}_0, Z)\, \pi_A(O_{1:K}, y|x, \mathcal{G}_1, F) \tag{3}$$

This structure directly resolves the limitations of Eq. 1: π_P and π_A become separately optimisable policies, each amenable to its own reward (resolving role entanglement); rewards can be assigned to $\mathcal{G}_1$, to individual O_k, and to y respectively, distributing supervision across the rollout (resolving sparsity); and $\mathcal{G}$ is an explicit structured object that the planner emits and the answerer consumes (resolving expressiveness, with the structure detailed in Sec. 3.3). Consequently the training objective decomposes into two role-specific objectives,

$$\begin{aligned} \mathcal{J}_P(\theta_P) &= \mathbb{E}_x[R_P(\mathcal{G}_1; x, Z)]\,, \\ \mathcal{J}_A(\theta_A) &= \mathbb{E}_x[R_A(O_{1:K}, y; x, \mathcal{G}_1)] \end{aligned} \tag{4}$$

optimised in two sequential RL stages with R_P and R_A representing the planner and answerer reward respectively.

3.2 Overview

Figure 1 illustrates the DecomposeR rollout, which is identical at training and inference time up to the application of gradients. Given a query x, the planner first emits an initial plan $\mathcal{G}_0$ in the form of a typed

DAG (the construction of which is detailed in Sec. 3.3). The environment $\mathcal{E}$ executes every search node in $\mathcal{G}_0$ and returns the observation set Z. The planner consumes Z and emits a revised plan $\mathcal{G}_1$ together with a small set of URLs F to be fetched; $\mathcal{E}$ then runs only the newly added or modified search nodes in $\mathcal{G}_1$ and fetches the URLs in F, reusing observations from unchanged search nodes. Finally, the answerer fills the aggregate nodes of $\mathcal{G}_1$ in topological waves and produces the final citation-grounded report at the answer node. The three design choices underlying this pipeline are justified below.

Why a typed DAG. A typed DAG with search-, aggregate-, and answer-type nodes makes cross-source evidence reuse and hierarchical synthesis explicit at the structural level, both of which linear lists and tree-structured plans cannot natively express. The structure further exposes well-defined properties– branch breadth, cross-branch integration, query distinctness– as addressable units that the planner reward can target directly (Sec. 3.4.2).

Why two-turn planning. $\mathcal{G}_0$ is generated from x alone and therefore reflects only the planner's parametric guess at what is worth investigating. Parametric knowledge frequently misaligns with what is actually retrievable on the open web: queries can return sparse, off-topic, or outdated snippets. The second turn closes the planning loop with retrieval reality, letting the planner drop ineffective subqueries, add new subqueries inspired by retrieved snippets, and concentrate full-page fetching on the URLs most likely to support the synthesis.

Why decouple planner and answerer. Collapsing planning and answering into one policy under a single trajectory-level reward conflates their failure modes: a low answer score cannot distinguish a weak plan executed faithfully from a strong plan executed poorly. By training the two roles sequentially, Planner RL optimises rewards that depend only on $\mathcal{G}_1$ and never on answerer output, so plan quality is isolated from execution noise. Answerer RL then trains against a fixed planner, so answerer credit is not confounded by drifting plan quality. Staging also reduces the non-stationarity inherent to jointly training two policies that act as each other's environment, leaving two stable single-policy optimisation problems.

3.3 Plan Representation

The plan $\mathcal{G}$ must satisfy three requirements: expose addressable structure for fine-grained reward design, capture cross-source evidence reuse and hierarchical synthesis, and admit efficient execution. Linear subquery lists provide only a flat sequence with no notion of branch reuse, and tree-structured subgoal plans forbid a node from being shared across multiple descendants. We therefore represent $\mathcal{G}$ as a typed directed acyclic graph, whose construction and execution we now detail.

Node types. $\mathcal{G}$ contains three node types. A search node $s \in \mathcal{N}_s$ carries a query string and a list of key points specifying what evidence the query should return. An aggregate node $a \in \mathcal{N}_a$ carries a synthesis brief consisting of a need statement and key points to be covered in its output. The terminal answer node v_{ans} carries the final synthesis brief; it has no outgoing edges, and exactly one such node exists per plan.

Edges and validity. A directed edge $u \to v$ declares that the output of u is consumed by v: edges from search nodes carry retrieved evidence, edges between aggregate nodes carry branch-level conclusions, and edges into v_{ans} carry the inputs to the final report. A validator enforces that $\mathcal{G}$ is a connected DAG with valid JSON syntax, valid node IDs, allowed node types, no cycles, and exactly one v_{ans}. Invalid graphs are rejected at the validity gate before any reward is computed (Sec. 3.4.2).

Topological-wave execution. Given a valid $\mathcal{G}_1$, the answerer executes it wave by wave. Wave W_ℓ contains all aggregate nodes whose parents have already been completed; all nodes in a wave are generated in a single model turn, exploiting the fact that same-wave aggregates are conditionally independent given their parents. After all aggregate nodes are filled, a final turn generates v_{ans} conditioned on its parent branches and the cumulative cited evidence. Topological execution preserves the dependency order implied by $\mathcal{G}$ while minimising the number of model turns required.

3.4 Staged Training

We train the planner and the answerer sequentially atop a shared Qwen3-8B backbone with role-conditioned prompts and role-specific LoRA adapters (Hu et al., 2021): a planner adapter is active when the model emits a graph, and an answerer adapter is active when the model emits aggregate or answer outputs. Training proceeds in three phases: a cold-start SFT phase that initialises both adapters from teacher trajectories, a planner RL phase that refines the planner adapter, and an answerer RL phase that refines the answerer adapter starting from a checkpoint with the planner-RL adapter merged in. Both RL phases use Group Relative Policy Optimization (GRPO) (Shao et al., 2024) with token-level masking that excludes environment observations from the policy gradient; the full GRPO objective and masking definitions are deferred to Appendix A.2. We describe each phase below, focusing on the DecomposeR-specific reward design.

3.4.1 Cold-Start SFT

We first perform supervised fine-tuning to teach the model the basic graph-emission protocol and wave-execution format, so that subsequent RL operates on a policy that already produces parseable outputs rather than wasting samples on protocol violations. A frontier teacher model (GPT-5-mini) generates complete planner–answerer trajectories for a held-out subset of training queries; each trajectory comprises planner turns (initial and revised graphs) and answerer turns (aggregate waves and final answer). We retain only trajectories that pass the graph validator and the answerer parser end-to-end, discarding those with malformed JSON, invalid node references, or schema-violating output. Surviving trajectories yield two streams of turn-level examples– planner turns under the planner system prompt, and answerer turns under the answerer system prompt– on which the planner and answerer adapters are jointly initialised.

3.4.2 Planner RL

The planner adapter is refined with a reward over three plan-quality dimensions: semantic coverage of the user's information need, retrieval quality of the search queries, and structural expressiveness of the DAG, rewarding each independently before combining them:

$$R_{\text{plan}} = \mathbf{1}[\text{Valid}(\mathcal{G})] \left(\lambda_{\text{cov}}\, C_{\text{rub}} + \lambda_{\text{search}}\, Q_{\text{search}} + \lambda_{\text{graph}}\, E_{\text{graph}}\right). \tag{5}$$

The hard validity gate $\mathbf{1}[\text{Valid}(\mathcal{G})]$ ensures that protocol correctness is established before quality signals take effect so invalid graphs receive zero reward regardless of their semantic content. All λ terms in this subsection are scalar reward weights.

Rubric coverage C_{rub}. For each training query, a frontier LLM generates a query-specific rubric $\mathcal{R} = \{r_u\}$. Each rubric item r_u describes content that the final response should cover. Rubric coverage measures how well the emitted plan anticipates these criteria before the final answer is written. For each rubric item, we compare its embedding to every non-answer plan-node field, namely `query`, `key_points`, and `need`. The item receives the strongest nonnegative semantic match anywhere in the plan, and C_{rub} averages these best-match scores across rubric items:

$$C_{\text{rub}} = \frac{1}{|\mathcal{R}|} \sum_{r_u \in \mathcal{R}} \text{BestMatch}(r_u, \mathcal{G}). \tag{6}$$

Here $\text{BestMatch}(r_u, \mathcal{G})$ is the best nonnegative embedding similarity between rubric item r_u and any non-answer plan-node field. Thus, C_{rub} is high when the planner has already included fields that semantically match the rubric items, even before the final answer node is generated.

Search quality Q_{search}. Search quality measures whether the search nodes target the right aspects and avoid redundancy:

$$Q_{\text{search}} = \lambda_{\text{search,cov}}\, C_{\text{search}} + \lambda_{\text{search,dist}}\, D_{\text{search}}. \tag{7}$$

Its coverage part, C_{search}, scores how well the emitted search-query strings match the rubric items. Its distinctness part, D_{search}, penalizes search queries that are close semantic paraphrases of one another, with the distinctness term set to zero when fewer than two search queries are emitted. Thus Q_{search} rewards search nodes that are both on-topic and nonredundant.

Figure 2: Planner RL reward dynamics. The black curve shows the smoothed planner reward and the grey trace shows raw rollout means. Structural diagnostics are reported in Appendix A.6.

Graph expressiveness E_{graph}. Graph expressiveness combines synthesis breadth B_{synth}, cross-branch integration I_{cross}, and search breadth B_{search}.

$$E_{\text{graph}} = \lambda_{\text{graph,synth}} B_{\text{synth}} + \lambda_{\text{graph,cross}} I_{\text{cross}} + \lambda_{\text{graph,search}} B_{\text{search}}. \tag{8}$$

Synthesis breadth rewards having multiple semantically distinct aggregate branches feed the answer node. Cross-branch integration rewards two structural patterns: search evidence reused by multiple aggregate nodes, and aggregate nodes that combine multiple parent inputs. Search breadth rewards using more than one search node, but with diminishing returns so the planner is not pushed toward unbounded graph growth.

Full mathematical definitions of all planner reward components are given in Appendix A.5.

3.4.3 Answerer RL

With the planner adapter fixed, the answerer adapter is refined to execute the plan faithfully and synthesise a high-quality final report. Graphs are generated by the planner-RL model and rolled out through the environment in advance, so the answerer trains on a stable distribution of plans rather than on a moving target. We decompose answerer quality into two dimensions: branch-level execution fidelity (measuring whether each aggregate node covers its declared key points) and global synthesis quality (measuring whether the final answer integrates the branches and satisfies the rubric) and combine them as

$$R_{\text{ans}} = \lambda_{\text{exec}} R_{\text{exec}} + \lambda_{\text{synth}} R_{\text{synth}} \tag{9}$$

All λ terms in this subsection are scalar reward weights. Parse failures receive partial credit computed over the successfully completed waves; this eliminates the binary reward cliff that destabilised early experiments, in which a single malformed node would zero out a trajectory whose mean baseline was well above zero.

Execution reward R_{exec}. Execution reward scores each aggregate node independently against its declared key points. Only aggregate nodes are scored, since search nodes are executed by $\mathcal{E}$ rather than by the policy. For each $v \in \mathcal{N}_a$, we combine an embedding-level and a lexical key-point coverage signal:

$$r_v = \lambda_{\text{exec,emb}} K_{\text{emb}}(v) + \lambda_{\text{exec,lex}} K_{\text{lex}}(v) \tag{10}$$

$K_{\text{emb}}(v)$ is a greedy bipartite matching score between the node's `key_points` embeddings and its output-sentence embeddings, weighted by recall; $K_{\text{lex}}(v)$ is the mean fraction of each key point's content tokens present in the output. The embedding signal captures semantic coverage, while the lexical signal grounds it deterministically. Per-node scores are aggregated with a shifted harmonic mean:

$$R_{\text{exec}} = \frac{|\mathcal{N}_a|}{\sum_{v \in \mathcal{N}_a} \frac{1}{1+r_v}} - 1 \in [0, 1] \tag{11}$$

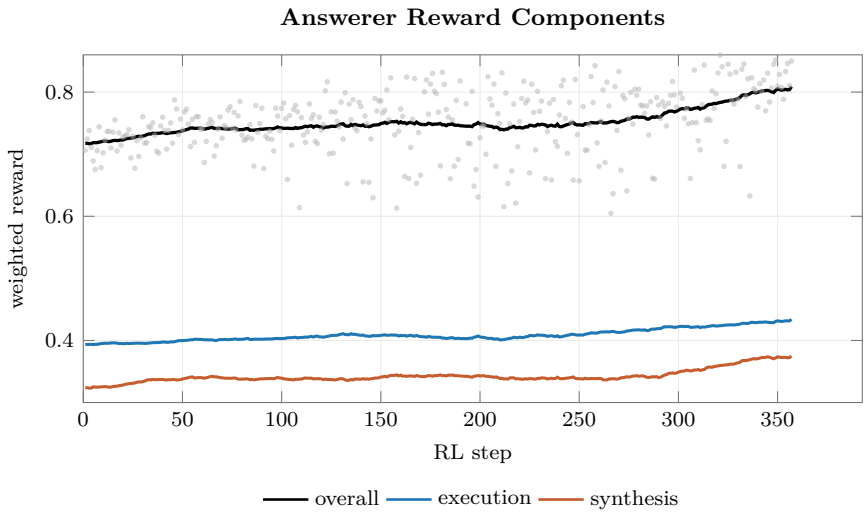

Figure 3: Answerer RL reward dynamics. The plot shows overall answerer reward together with its execution and synthesis contributions. Length diagnostics are reported in Appendix A.6.

The shifted harmonic mean maps all-zero to 0 and all-one to 1.

Synthesis reward R_{synth}. Synthesis reward targets the final answer rather than individual branches:

$$R_{\text{synth}} = \lambda_{\text{synth,judge}} J_{\text{rub}} + \lambda_{\text{synth,util}} U_{\text{branch}}. \tag{12}$$

J_{rub} is the mean judge score across rubric items from an external LLM on a five-point scale. U_{branch} is the mean fraction of each branch's content tokens that appear in the final answer; it provides a deterministic signal that the model is actually consuming its intermediate branch outputs.

Full mathematical definitions of all answerer reward components are given in Appendix A.5.

4 Experiments

4.1 Experimental Setup

Benchmarks and metrics. We evaluate DecomposeR on three long-form deep-research benchmarks spanning different domains. **DeepResearchBench** provides expert-written multi-domain research tasks that require web exploration, citation-backed synthesis, and report-quality judgment. **ResearchQA-Mini** evaluates scholarly question answering with survey-derived research questions and rubric items. **HealthBench** evaluates responses to health-related conversations against physician-informed rubric criteria, testing evidence synthesis in a high-stakes domain. For all three benchmarks we report the official rubric- or judge-based score; each metric captures coverage, factual correctness, and response quality on a scale where higher is better. Appendix A.4 provides benchmark-specific evaluation details.

Base model. We build DecomposeR on Qwen3-8B (Qwen Team, 2025) with two LoRA adapters (Hu et al., 2021): one for the planner role and one for the answerer role. All ablation variants use this same backbone with the same SFT initialization, so that comparisons across variants isolate the effect of the studied design choice.

Baselines. We compare DecomposeR-8B against two broad types of systems: *(i) Commercial deep-research systems*: OpenAI (o4-mini) Deep Research, Gemini Deep Research and Perplexity Deep Research, accessed through their official APIs. *(ii) Open deep-research and search agents*: WebThinker (QwQ-32B and 32B-DPO), Tongyi DeepResearch-30B, WebExplorer-8B, Search-R1, ASearcher-Web-7B and Qwen3-8B equipped with search. We additionally report a DecomposeR-8B (SFT only) variant to isolate the effect of the two RL stages.

Training data. We obtain a filtered pool of roughly 4,000 long-form English prompts/queries across OpenScholar, SearchArena, and ScholarQA sources. We use 2,325 queries for cold-start SFT, converted from teacher planner–answerer trajectories into 10,532 turn-level supervision examples. For RL, both planner RL and answerer RL use the same 1,432-query prepared set, with planner graphs, search results, fetched pages, rubrics, and citation catalogs attached to each query artifact.

Training protocol. Both RL stages use GRPO (Shao et al., 2024) with verl (Sheng et al., 2024) and vLLM (Kwon et al., 2023) for training and rollout generation respectively. The environment uses Jina Search for search-node execution (up to five results per node) and Serper for URL fetching (up to four URLs per revision turn). Full GRPO details, hyperparameters, and prompt templates are in Appendix A. A central design decision is that **reward weights are specified once and kept fixed** across the main experiment and every ablation variant reported in this section. The reward specification uses no benchmark-specific weight calibration. Consequently, performance differences across variants reflect the design choices being studied rather than reward retuning. Reward weights used are reported in full in Appendix A.5.

4.2 Main Results

Overall performance. Table 1 shows that DecomposeR-8B (SFT+RL) is the strongest open model across the three long-form benchmarks compared against models of similar size, achieving 41.8 on DeepResearchBench, 42.0 on HealthBench, and 71.4 on ResearchQA-Mini. Compared with Qwen3-8B + Search, DecomposeR improves by 7.4 points on DeepResearchBench, 22.2 points on HealthBench, and 14.4 points on ResearchQA-Mini, despite using the same backbone scale. It also substantially outperforms WebExplorer-8B and other RL-trained search agents such as Search-R1-7B and ASearcher-Web-7B indicating that the improvement is not simply due to giving an 8B model access to search, but training it to organize retrieval and synthesis through the DecomposeR framework.

RL improves substantially over SFT. The SFT-only model already benefits from learning the typed graph and wave-execution protocol, but the two RL stages provide large additional gains. The improvement is largest on ResearchQA-Mini, where rubric-style coverage and synthesis are central, and remains positive on HealthBench, where domain sensitivity makes unsupported synthesis more costly. These gains suggest that SFT is sufficient to initialize the graph-emission and wave-execution protocol, but that staged RL is needed for the planner and answerer to use this protocol effectively.

Comparison with larger open systems. Among open deep-research systems, DecomposeR is the best on average and is competitive with or stronger than substantially larger models on individual benchmarks. It outperforms Tongyi DeepResearch-30B-A3B on DeepResearchBench and ResearchQA-Mini and exceeds WebThinker-32B-DPO on DeepResearchBench and HealthBench, while remaining close on ResearchQA-Mini. This is notable because those systems rely on much larger backbones and/or more carefully-engineered inference behavior.

4.3 Staged Training is Necessary

The two experiments in this section answer two distinct questions: whether staged training outperforms joint training (B1), and whether the improvement actually traces back to better plans rather than to the answerer compensating for weak plans (B2). Both halves of Table 2 share the same fixed reward weights as the main experiment.

B1: Staged vs. Joint RL. We construct a Joint RL baseline that shares with DecomposeR the same Qwen3-8B backbone, separate planner and answerer LoRA adapters, SFT initialization, and typed DAG plan representation. The variants differ only in training schedule and reward assignment. In Joint RL, the planner and answerer adapters are updated simultaneously from a single trajectory-level reward computed from the final answer. This setup isolates the effect of staged, role-specific optimization from changes in model architecture or plan representation. The top half of Table 2 shows that staged training outperforms joint training by 3.7–4.7 points across benchmarks. The gap is largest on DeepResearchBench, whose general-domain queries produce the longest rollouts and therefore suffer most from sparse trajectory-end credit assignment.

Table 1: Main results on three long-form deep-research benchmarks. DecomposeR-8B (SFT+RL) refers to the full model trained with both planner-RL and answerer-RL stages. Entries marked "–" indicate that no public result was available or that running the evaluation was prohibitively expensive.

System	Family	DeepResearchBench	HealthBench	ResearchQA-Mini	Average
Commercial deep-research systems					
OpenAI Deep Research	DR agent	46.9	54.0	79.1	60.0
Gemini Deep Research	DR agent	48.9	52.5	68.8	56.7
Perplexity Deep Research	DR agent	42.7	–	75.1	–
GPT-5 + Search	LLM+search	50.7	–	78.2	–
Open deep-research or search agents					
WebThinker QwQ-32B	Open DR	37.9	36.5	72.3	48.9
WebThinker-32B-DPO	Open DR	40.6	39.4	74.1	51.4
Tongyi DeepResearch-30B-A3B	Open DR	40.6	45.9	66.7	51.1
WebExplorer-8B	Open search	36.7	34.0	65.5	45.4
Search-R1-7B	Open search	9.7	1.1	28.4	13.1
ASearcher-Web-7B	Open Search	10.0	-13.0	22.5	6.5
Qwen3-8B + Search	LLM+search	34.4	19.8	57.0	37.1
DecomposeR-8B (SFT only)	Ours	36.3	39.1	63.8	46.4
DecomposeR-8B (SFT + RL)	Ours	**41.8**	**42.0**	**71.4**	**51.7**

Table 2: Validating staged training. B1 compares RL-pipeline variants trained on the same SFT model. B2 isolates planner quality by holding the answerer fixed and varying only the planner.

Variant	DeepResearchBench	HealthBench	ResearchQA-Mini
B1: Same architecture, different training schedule			
Joint RL	37.1	38.3	66.9
Staged RL (Ours)	**41.8**	**42.0**	**71.4**
B2: Different planners, fixed external answerer (GPT-4.1)			
GPT-5 (prompt only)	39.5	40.3	68.5
SFT-only planner	35.4	36.0	61.3
Joint-RL planner	37.5	37.3	65.5
DecomposeR planner	**39.8**	**39.7**	**69.4**

B2: Planner quality with a fixed external answerer. A natural concern is that the gains in B1 might come from the answerer learning to compensate for weak plans rather than from the planner producing better plans. We address this by holding the answerer fixed and varying only the planner. Specifically, we use GPT-4.1 with a standardized execution prompt as an external answerer and feed it plans from four planners: GPT-5 with a planning prompt (strong training-free reference), the SFT-only planner, the Joint-RL planner from B1, and the DecomposeR planner. The same external answerer executes all four plan distributions, so differences in final answer quality reflect only differences in plan quality. The bottom half of Table 2 shows that the DecomposeR planner produces plans whose downstream answer quality exceeds the SFT-only and Joint-RL planners considerably and closely trails or exceeds the GPT-5 planning baseline despite using a substantially smaller backbone. Combined with B1, this confirms that the gains from staged training trace back to the planner, not to answerer compensation.

4.4 Reward Component Ablations

We ablate the planner and answerer reward components separately, removing one component at a time from the full reward and retraining from the SFT checkpoint with the same schedule, data, and fixed weights on the remaining components. Results are reported in Table 3.

Table 3: Leave-one-out ablation of reward components. All variants keep the fixed reward weights on remaining components unchanged.

Variant	DeepResearchBench	HealthBench	ResearchQA-Mini
C1: Planner reward components			
Full R_{plan}	**41.8**	**42.0**	**71.4**
w/o C_{rub}	36.3	37.9	65.2
w/o Q_{search}	38.9	39.7	68.8
w/o E_{graph}	34.1	35.8	66.3
C2: Answerer reward components			
Full R_{ans}	**41.8**	**42.0**	**71.4**
w/o R_{exec}	36.1	31.9	62.8
w/o J_{rub}	38.0	39.3	69.8
w/o U_{branch}	40.5	41.3	70.3

C1: Planner reward components. The top half of Table 3 reports leave-one-out ablations of the three top-level planner reward terms. Removing rubric coverage C_{rub} causes the largest drop on ResearchQA-Mini, whose rubric-based evaluations directly reward semantic coverage of specific aspects. Removing search quality Q_{search} degrades all three benchmarks, falling from 51.7 to 49.1 on average, confirming that this term shapes retrieval coverage rather than only acting as a noise term. Removing graph expressiveness E_{graph} has a sizeable effect on final answer scores and causes graphs to collapse toward less integrated, simpler structures. This contrasts with the full planner-RL diagnostic in Figure 4, where edge count grows faster than node count, indicating increasing cross-branch reuse.

C2: Answerer reward components. The bottom half of Table 3 reports the same ablation for the answerer reward. Removing per-aggregate execution reward R_{exec} causes branch-level key-point coverage to drop substantially, resulting in the largest drop-off across all answerer sub-components. Removing the rubric judge J_{rub} reduces final answer quality on all three benchmarks. The branch utilization term U_{branch} has the smallest standalone effect on benchmark scores, but when removed the content overlap between the final answer and intermediate branch outputs drops from 0.82 to 0.61, indicating that without U_{branch} the answerer increasingly ignores its own branch outputs and re-derives the final report from scratch.

4.5 Plan Representation and Revision Ablations

This ablation complements the reward-component analysis in Section 4.4 by isolating two structural choices in the planner: whether the plan is a typed DAG rather than a weaker linear or tree structure, and whether the planner can revise the plan after seeing search results. Results are reported in Table 4.

D1: Plan structure ablation. We replace the typed DAG with two alternative plan representations while retaining the rest of the DecomposeR training pipeline. **Linear**: the plan consists of search nodes feeding directly into the answer node with no aggregate layer, approximating the parallel-subquery structures used by ParallelSearch and SubSearch; the graph expressiveness reward E_{graph} is removed because there is no DAG structure to score, leaving C_{rub} and Q_{search} active. **Tree**: aggregate nodes are restricted to at most one parent, approximating hierarchical outline structures of WebWeaver-style systems; the structural rewards remain active but r_{int} in I_{cross} is necessarily zero. The top half of Table 4 shows that the typed DAG outperforms both Linear and Tree on every benchmark, with the largest gap on tasks that require synthesizing evidence across multiple thematic branches.

D2: Single-turn vs. two-turn planning. We further ablate the two-turn planner protocol by training a single-turn variant in which the planner emits $\mathcal{G}_0$ and immediately commits to it, with no opportunity to revise after seeing search results. The bottom half of Table 4 shows that two-turn planning improves over

Table 4: Plan representation ablations. D1 replaces the typed DAG with a linear or tree structure. D2 disables the revision turn that follows initial search.

Variant	DeepResearchBench	HealthBench	ResearchQA-Mini
D1: Plan structure			
Linear	34.2	31.5	57.8
Tree	38.6	39.8	65.3
Typed DAG (Ours)	**41.8**	**42.0**	**71.4**
D2: Planner revision			
Single-turn planning	35.1	36.7	67.5
Two-turn planning (Ours)	**41.8**	**42.0**	**71.4**

single-turn planning on all three benchmarks, consistent with the intuition that an initial plan generated from parametric knowledge alone benefits from being adjusted to what is actually retrievable.

4.6 Case Study

Appendix C gives a representative DecomposeR rollout, including the pre- and post-revision graphs in Figures 15 and 17, the turn flow in Table 9, and the revised graph summary in Table 10. The example highlights framework-specific behavior to look for: the initial decomposition is revised after search feedback; added or modified search nodes and fetch requests change the evidence available to the answerer; search evidence can feed multiple downstream aggregates; and aggregate outputs are executed in topological waves before being combined by the terminal answer node.

5 Conclusion

We introduced DecomposeR, a planner-centric framework for deep research that represents plans as typed DAGs and trains planner and answerer roles in separate RL stages. By assigning reward to explicit graph components, DecomposeR improves credit assignment for the planning process. The planner RL run shows strong gains in reward, validity, search breadth, aggregate branching, and cross-branch structure, supporting the core hypothesis that structured plans are a useful optimization target for deep research systems. The answerer RL run confirms that execution fidelity and synthesis quality are separable learning objectives.

6 Limitations

The current system fixes the plan at graph revision time and does not support iterative replanning after the answerer begins execution. In principle, partial evidence from early aggregate nodes could inform later search decisions, but this would require variable-length rollouts that make GRPO credit assignment substantially harder. The two-turn bounded planner is a deliberate tradeoff that preserves RL tractability; extending to adaptive replanning is a natural future direction.

The reward design uses fixed weights across several planner and answerer components. We keep these weights fixed for all experiments to avoid benchmark-specific tuning, but this may not be optimal across domains. In addition, several reward terms rely on embedding-based rubric or key-point matching, which can over-credit semantically similar but incomplete content.

References

Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Quiñonero-Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, Johannes Heidecke, and Karan Singhal. Healthbench: Evaluating large language models towards improved human health, 2025. URL https://arxiv.org/abs/2505.08775.

Akari Asai, Jacqueline He, Rulin Shao, Weijia Shi, Amanpreet Singh, Joseph Chee Chang, Kyle Lo, Luca Soldaini, Sergey Feldman, Mike D'arcy, David Wadden, Matt Latzke, Minyang Tian, Pan Ji, Shengyan Liu, Hao Tong, Bohao Wu, Yanyu Xiong, Luke Zettlemoyer, Graham Neubig, Dan Weld, Doug Downey, Wen tau Yih, Pang Wei Koh, and Hannaneh Hajishirzi. Openscholar: Synthesizing scientific literature with retrieval-augmented lms, 2024a. URL https://arxiv.org/abs/2411.14199.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, volume 2024, pp. 9112–9141, 2024b.

Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Chenzheng Zhu, Haofen Wang, Jeff Pan, Wen Zhang, Huajun Chen, Fan Yang, et al. Learning to reason with search for llms via reinforcement learning. *Advances in Neural Information Processing Systems*, 38:85287–85307, 2026.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents. *arXiv preprint arXiv:2506.11763*, 2025.

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

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7969–7992, 2023.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O. Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. In *Proceedings of the Conference on Language Modeling*, 2025.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *International Conference on Learning Representations*, 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

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. Websailor: Navigating super-human reasoning for web agent, 2025a. URL https://arxiv.org/abs/2507.02592.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. Webthinker: Empowering large reasoning models with deep research capability. *Advances in Neural Information Processing Systems*, 38:120091–120131, 2026.

Zijian Li, Xin Guan, Bo Zhang, Shen Huang, Houquan Zhou, Shaopeng Lai, Ming Yan, Yong Jiang, Pengjun Xie, Fei Huang, Jun Zhang, and Jingren Zhou. Webweaver: Structuring web-scale evidence with dynamic outlines for open-ended deep research. *arXiv preprint arXiv:2509.13312*, 2025b.

Junteng Liu, Yunji Li, Chi Zhang, Chengyu Du, Qidi Xu, Jiayuan Song, Jingyang Li, Aili Chen, Ke Ji, Weiyu Cheng, Zijia Wu, Zhengmao Zhu, Wenhu Chen, Pengyu Zhao, and Junxian He. Webexplorer: Explore and evolve for training long-horizon web agents. *arXiv preprint arXiv:2509.06501*, 2025.

Jianbiao Mei, Tao Hu, Daocheng Fu, Licheng Wen, Xuemeng Yang, Rong Wu, Pinlong Cai, Xinyu Cai, Xing Gao, Yu Yang, Chengjun Xie, Botian Shi, Yong Liu, and Yu Qiao. O^2-searcher: A searching-based agent model for open-domain open-ended question answering. *arXiv preprint arXiv:2505.16582*, 2025.

OpenAI. Introducing deep research. `https://openai.com/index/introducing-deep-research/`, 2025.

Perplexity Team. Introducing perplexity deep research. `https://www.perplexity.ai/hub/blog/introducing-perplexity-deep-research`, 2025.

Qwen Team. Qwen technical reports. `https://qwenlm.github.io/`, 2025.

Sentence Transformers. sentence-transformers/all-MiniLM-L6-v2. `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`, 2021. Hugging Face model card.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng et al. Verl: Volcano engine reinforcement learning for llms. *arXiv preprint arXiv:2409.19256*, 2024.

Amanpreet Singh, Joseph Chee Chang, Dany Haddad, Aakanksha Naik, Jena D. Hwang, Rodney Kinney, Daniel S. Weld, Doug Downey, and Sergey Feldman. AI2 scholar QA: Organized literature synthesis with attribution. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 513–523, 2025.

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

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023.

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

Shu Zhao, Tan Yu, Anbang Xu, Japinder Singh, Aaditya Shukla, and Rama Akkiraju. Parallelsearch: Train your llms to decompose query and search sub-queries in parallel with reinforcement learning. *arXiv preprint arXiv:2508.09303*, 2025.

Appendix

A Training Details

A.1 Training Data Sources

Table 5 below gives the exact source breakdown for the filtered long-form training pool and the artifacts derived from it. The active training sets contain 2,325 SFT queries, converted into 10,532 supervised turns, and a shared 1,432-query RL set used by both planner RL and answerer RL.

Table 5: Training data source breakdown. "SFT used" counts planner–answerer teacher trajectories used for cold-start SFT. "RL used" counts the shared prepared query set used by both planner RL and answerer RL.

Source	Pool	SFT used	RL used
OpenScholar	2,577	1,554	897
SearchArena	1,039	558	403
ScholarQA	384	213	132
Total queries/artifacts	4,000	2,325	1,432

A.2 Masked GRPO Objective

This subsection expands the Group Relative Policy Optimization (GRPO) training protocol summarized in Section 4.1 (Shao et al., 2024). Search-RL systems such as Search-R1 and ReSearch adapt GRPO to tool environments by treating retrieved text as environment output and masking it out of the policy-gradient loss (Jin et al., 2025; Chen et al., 2026); O_2-Searcher further uses GRPO with composite rewards for open-ended search tasks (Mei et al., 2025). DecomposeR follows the same separation principle, but the model-generated objects are typed planner graphs and answerer node reports rather than inline search traces.

Masked GRPO objective. For each input query or prepared answerer artifact x, the old policy samples a group of G rollouts $\{\tau_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot|x)$. Each rollout receives a scalar stage reward R_i: $R_i = R_{\text{plan}}$ for planner RL and $R_i = R_{\text{ans}}$ for answerer RL. We use the full clipped GRPO objective of Shao et al. (2024), with the token average restricted to the model-generated assistant tokens:

$$
\begin{aligned}
\mathcal{J}_{\text{GRPO}}(\theta) &= \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|\mathcal{M}_i|}\sum_{t\in\mathcal{M}_i}\left\{\min\Big(\rho_{i,t}\widehat{A}_i, \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon)\widehat{A}_i\Big) - \beta\,\widehat{D}_{\text{KL}}^{i,t}\right\}\right], \\
\rho_{i,t} &= \frac{\pi_\theta(y_{i,t}|h_{i,t})}{\pi_{\theta_{\text{old}}}(y_{i,t}|h_{i,t})}, \qquad \widehat{A}_i = \frac{R_i - \frac{1}{G}\sum_{j=1}^{G} R_j}{\text{std}(\{R_j\}_{j=1}^{G}) + \delta}, \\
\widehat{D}_{\text{KL}}^{i,t} &= \frac{\pi_{\text{ref}}(y_{i,t}|h_{i,t})}{\pi_\theta(y_{i,t}|h_{i,t})} - \log\left(\frac{\pi_{\text{ref}}(y_{i,t}|h_{i,t})}{\pi_\theta(y_{i,t}|h_{i,t})}\right) - 1.
\end{aligned}
\tag{13}
$$

Here $h_{i,t}$ is the full dialogue context before token $y_{i,t}$, ϵ is the PPO-style clip range, β is the KL coefficient, and δ is a small numerical constant. The term $\widehat{D}_{\text{KL}}^{i,t}$ is the positive KL estimator used by GRPO; unlike RLHF formulations that add a per-token KL penalty into the reward, this form regularizes the policy directly in the objective. No learned critic or value head is used; the group mean and standard deviation provide the baseline and scale. Since our rewards are trajectory-level stage rewards, the same normalized advantage is assigned to every unmasked policy token in the rollout.

The mask $\mathcal{M}_i$ contains only policy-generated assistant tokens. For planner RL, this includes the planner `<think>` tokens, graph JSON, revision JSON, and fetch-request JSON. For answerer RL, this includes the answerer `<think>` tokens and XML `<node>` blocks for aggregate waves and the final answer. System prompts, user payloads, Jina search results, Serper fetch contents, compact dependency payloads, and citation catalogs

Table 6: Cold-start SFT hyper-parameters.

Setting	Cold-start SFT
Maximum sequence length	16,384 tokens
Epochs	2
Learning rate	1.0×10^{-5}
Batching	per-device batch size 1; gradient accumulation 8
Temperature for rollout sampling	1.0
Optimizer and schedule	fused AdamW, cosine schedule
Weight decay	0.0
Precision and memory	BF16, gradient checkpointing, DeepSpeed ZeRO-2

Table 7: GRPO hyper-parameters used for the two RL stages.

Setting	Planner RL	Answerer RL
Starting checkpoint	Turn-level SFT merged checkpoint	Planner-RL merged checkpoint
Rollout backend	VERL with vLLM rollouts	VERL with vLLM rollouts
Model dtype	BF16	BF16
LoRA rank	64	64
LoRA alpha	128	128
Unique prompts per batch	8	4
Rollouts per prompt (group size)	6	6
Mini-batch size	4	2
Mini-batches per GRPO step	2	2
Inner epochs per batch	1	1
Max prompt length in tokens	4,096	8,192
Max response length in tokens	8,192	16,384
Max tokens in packed sequence	12,288	28,672
Temperature for sampling rollouts	1.0	1.0
Top-p for sampling rollouts	1.0	1.0
KL coefficient β	0.001	0.005
Learning rate schedule	constant, no warmup	constant, no warmup
Learning rate	4.0×10^{-6}	4.0×10^{-6}
Weight decay	0.0	0.0
Environment	Jina Search, up to five results per search node; Offline Serper fetch, up to four URLs	Offline prepared planner graphs, search results, fetched pages, and citation catalog

are conditioned on but excluded from the policy-gradient and KL terms. This is the DecomposeR analogue of retrieved-token masking in inline search-agent rollouts.

A.3 Training Hyper-parameters

Tables 6 and 7 report the cold-start SFT and RL hyper-parameters used in the reported run. All training stages were run on a single node with 8 x NVIDIA RTX PRO 6000 GPUs, each with 96GB of memory.

A.4 Benchmark Evaluation Details

DeepResearchBench. We evaluated all 100 DeepResearchBench prompts, consisting of 50 Chinese and 50 English tasks from the official query file. Following the benchmark protocol, we generated one cited long-form report per prompt, ran the official RACE and FACT evaluation pipelines separately for the Chinese, English, and combined outputs. RACE was judged with the official DeepResearchBench LLM judge using `openai/gpt-5.5`, while the auxiliary FACT pipeline used `openai/gpt-5.4-mini`; the main results table reports the combined RACE score.

Table 8: All reward weights used in the planner and answerer RL stages.

Symbol	Description	Value
	Planner top-level (Eq. 5)	
λ_{cov}	Rubric coverage weight	0.40
λ_{search}	Search quality weight	0.30
λ_{graph}	Graph expressiveness weight	0.30
	Search quality sub-weights	
$\lambda_{\text{search,cov}}$	Search rubric coverage	0.50
$\lambda_{\text{search,dist}}$	Search distinctness	0.50
	Graph expressiveness sub-weights	
$\lambda_{\text{graph,synth}}$	Synthesis breadth	0.40
$\lambda_{\text{graph,cross}}$	Cross-branch integration	0.40
$\lambda_{\text{graph,search}}$	Search breadth	0.20
	Answerer top-level (Eq. 9)	
λ_{exec}	Execution reward weight	0.50
λ_{synth}	Synthesis reward weight	0.50
	Execution sub-weights	
$\lambda_{\text{exec,emb}}$	Embedding key-point match	0.50
$\lambda_{\text{exec,lex}}$	Lexical key-point recall	0.50
	Synthesis sub-weights	
$\lambda_{\text{synth,judge}}$	LLM final-answer rubric judge	0.50
$\lambda_{\text{synth,util}}$	Deterministic branch-token utilization	0.50

ResearchQA-Mini. We evaluated the full 776-question `test_mini` split of ResearchQA-Mini. The generated reports were exported in the expected ResearchQA response-map format and scored with the official recommended rubric-coverage judge used by the evaluation wrapper, `gpt-4.1-mini`.

HealthBench. We evaluated 500 samples sampled from the official HealthBench dataset, including 100 prompts from the hard subset under a hard-sampling fraction of 20%, consistent with the original ratio. Generated completions were attached back to the original HealthBench rows and graded with the official `simple-evals` HealthBench rubric implementation using the recommended `gpt-4.1-2025-04-14` as the grader.

A.5 Planner and Answerer Reward Details

Table 8 separates weights by the object they score. Planner terms are applied to the revised graph before any final answer is written, while answerer terms are applied to generated node reports and the terminal answer under a fixed graph. Throughout this subsection, $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ denotes the revised planner graph, $\mathcal{N}_s \subset \mathcal{N}$ denotes its search nodes, $\mathcal{N}_a \subset \mathcal{N}$ denotes its aggregate nodes, and v_{ans} denotes the terminal answer node. The function $\phi(\cdot)$ is the sentence-embedding encoder used for reward-side semantic matching, $\cos(\cdot,\cdot)$ is cosine similarity in that embedding space, and $\max(0, \cos(\cdot,\cdot))$ clamps negative similarities to zero.

Planner-stage reward. The planner rollout consists of an initial graph, environment search results, and a planner revision that emits the final graph and fetch requests. Only the revised graph $\mathcal{G}$ is rewarded. Invalid graphs receive zero reward:

$$R_{\text{plan}} = \mathbf{1}[\text{Valid}(\mathcal{G})]\,(0.40\,C_{\text{rub}} + 0.30\,Q_{\text{search}} + 0.30\,E_{\text{graph}})\,. \tag{14}$$

Here $\mathbf{1}[\text{Valid}(\mathcal{G})]$ is one only when the revised graph parses and satisfies the typed-DAG constraints; otherwise the planner reward is zero. The three terms correspond to distinct planner responsibilities: C_{rub} checks whether the plan creates places for the rubric content to be answered later, Q_{search} checks whether the search nodes expose useful and non-duplicative retrieval routes, and E_{graph} checks whether the graph contains enough reusable synthesis structure for the answerer to execute. Let $\text{fields}(n)$ be the non-empty `query`,

`key_points`, and `need` fields of a non-answer node n. Rubric coverage C_{rub} is the average best match from each rubric item to any searchable or aggregate plan field:

$$C_{\text{rub}} = \frac{1}{|\mathcal{R}|} \sum_{r_u \in \mathcal{R}} \max_{n \in \mathcal{N} \setminus \{v_{\text{ans}}\}} \max_{c \in \text{fields}(n)} \max(0, \cos(\phi(r_u), \phi(c))). \tag{15}$$

Here r_u is one rubric item, c is one candidate text field from a non-answer node, and the inner clamp gives zero credit to negative cosine similarities. Search quality is

$$Q_{\text{search}} = 0.50\, C_{\text{search}} + 0.50\, D_{\text{search}}, \tag{16}$$

where C_{search} measures rubric coverage by search queries and D_{search} measures inter-query distinctness. Let q_s be the query string emitted by search node $s \in \mathcal{N}_s$. Search coverage is

$$C_{\text{search}} = \frac{1}{|\mathcal{R}|} \sum_{r_u \in \mathcal{R}} \max_{s \in \mathcal{N}_s} \max(0, \cos(\phi(r_u), \phi(q_s))). \tag{17}$$

Let q_i and q_j be two emitted search queries. The distinctness term D_{search} is one minus the mean clamped cosine similarity over unordered pairs of search queries:

$$D_{\text{search}} = 1 - \frac{2}{|\mathcal{N}_s|(|\mathcal{N}_s| - 1)} \sum_{i<j} \max(0, \cos(\phi(q_i), \phi(q_j))). \tag{18}$$

When there are fewer than two search queries, D_{search} is set to zero. Thus C_{search} rewards query-level recall of important facets, while D_{search} discourages multiple search nodes from becoming paraphrases of the same retrieval request.

Graph expressiveness is

$$E_{\text{graph}} = 0.40\, B_{\text{synth}} + 0.40\, I_{\text{cross}} + 0.20\, B_{\text{search}}. \tag{19}$$

All three subterms use the diminishing-returns function $f(k) = \max(0, 1 - 1/\max(k, 1))$. Let $P_{\text{ans}} = \{a \in \mathcal{N}_a : (a, v_{\text{ans}}) \in \mathcal{E}\}$ be the set of aggregate parents of the answer node, and let d_{par} be one minus the mean pairwise normalized cosine similarity among their node texts. Then

$$B_{\text{synth}} = f(|P_{\text{ans}}|)\, d_{\text{par}}, \tag{20}$$

$$I_{\text{cross}} = \tfrac{1}{2}(r_{\text{fan}} + r_{\text{int}}), \tag{21}$$

$$B_{\text{search}} = f(|\mathcal{N}_s|), \tag{22}$$

where B_{synth} measures how many distinct aggregate branches feed the answer, I_{cross} measures cross-branch connectivity, and B_{search} measures search breadth. The term r_{fan} is the fraction of search nodes with at least two direct downstream children, so it rewards evidence reuse across branches. The term r_{int} is the fraction of aggregate nodes with at least two parent inputs, so it rewards synthesis nodes that combine evidence or upstream conclusions. The diminishing-returns function gives most of its gain when moving from one to two useful branches or search nodes, then saturates as counts grow; the diversity multiplier d_{par} prevents duplicate answer-parent branches from increasing B_{synth} cheaply.

Answerer-stage reward. The answerer rollout executes the fixed planner graph in topological aggregate waves and then writes the terminal answer node. For a fully parsed trajectory,

$$R_{\text{ans}} = 0.50\, R_{\text{exec}} + 0.50\, R_{\text{synth}}. \tag{23}$$

The execution term is local to aggregate nodes: it asks whether each branch report fulfilled the key points declared by the planner. The synthesis term is reserved for the terminal answer, so missing branch coverage and poor final integration affect different parts of the reward. Only aggregate nodes are scored for execution. For each aggregate node v with key points P_v and generated node text o_v,

$$r_v = 0.50\, K_{\text{emb}}(v) + 0.50\, K_{\text{lex}}(v). \tag{24}$$

A high r_v therefore means that a node report covers its assigned content targets, not merely that it is fluent or long. Here P_v is the set of planner-declared key points for aggregate node v, and o_v is the generated XML node text for that aggregate. The embedding term $K_{\text{emb}}(v)$ is the greedy one-to-one matching score between key points $p \in P_v$ and output sentences s from o_v, normalized by the number of key points:

$$K_{\text{emb}}(v) = \frac{1}{|P_v|} \sum_{(p,s)\in \text{match}(v)} \max(0, \cos(\phi(p), \phi(s))). \tag{25}$$

The lexical term strips inline citations and averages key-point token recall. Here $\text{tok}(\cdot)$ denotes the set of normalized content tokens after citation removal:

$$K_{\text{lex}}(v) = \frac{1}{|P_v|} \sum_{p\in P_v} \frac{|\text{tok}(p) \cap \text{tok}(o_v)|}{|\text{tok}(p)|}. \tag{26}$$

The embedding term gives credit for semantic coverage of key points, while the lexical term provides a deterministic guard against high semantic similarity without concrete key-point content. Node scores are combined with the shifted harmonic mean

$$R_{\text{exec}} = \frac{|\mathcal{N}_a|}{\sum_{v\in\mathcal{N}_a} \frac{1}{1+r_v}} - 1. \tag{27}$$

This preserves sensitivity to weak branches without turning one failed branch into a total zero reward. We use the shifted harmonic mean rather than a min operator because early experiments with a hard minimum made GRPO advantage estimates unstable under small group sizes.

Synthesis reward combines an LLM rubric judge and deterministic branch utilization:

$$R_{\text{synth}} = 0.50\, J_{\text{rub}} + 0.50\, U_{\text{branch}}. \tag{28}$$

These two subterms separate external answer quality from structural fidelity: J_{rub} measures whether the final report satisfies the query-level rubric, while U_{branch} checks whether the final report actually uses the branch reports produced during graph execution. The judge term J_{rub} is the mean of five-level scores $\{0, 0.25, 0.5, 0.75, 1\}$ returned by the configured judge (`gpt-5-mini` in the reported run). The five-level scale is used because coarser three-level scales merge adjacent rubric-satisfaction levels, while broader numeric scales introduce more calibration noise across queries. Branch utilization is

$$U_{\text{branch}} = \frac{1}{|B|} \sum_{b\in B} \frac{|\text{tok}(b) \cap \text{tok}(y)|}{|\text{tok}(b)|}, \tag{29}$$

where B is the set of aggregate branch outputs and y is the final answer, both after stripping inline citations. Each branch output $b \in B$ is one completed aggregate-node report. This term is deterministic and targets the failure mode in which the answerer ignores its completed branch reports and writes the final answer directly from parametric knowledge or retrieved evidence.

All reward-side semantic matching uses the `sentence-transformers/all-MiniLM-L6-v2` embedding model (Sentence Transformers, 2021).

Partial credit for parse failures. Planner parse or graph-validity failures receive zero because the planner reward is defined on a valid typed DAG. Answerer parse failures are handled differently: if some aggregate waves have already completed, missing aggregate nodes are assigned zero node scores and the rollout receives $0.50\, R_{\text{exec}}$ with $R_{\text{synth}} = 0$; if the final answer fails to parse after all aggregate waves succeed, it receives the same execution credit with zero synthesis. If no aggregate node is successfully parsed, the reward remains zero. This change was used because fully zeroing a long answerer rollout after a late XML parse error produced unstable GRPO advantages despite useful completed branch reports.

A.6 Training Diagnostics

Figures 4 and 5 report the two training diagnostics omitted from the main text for space. The planner graph curve shows that planner RL does not merely improve scalar reward: it also increases typed graph structure, with edge count growing faster than node count as the model learns to reuse evidence across branches. The answerer length curve shows that responses become longer during answerer RL. We interpret this as a consequence of key-point coverage pressure rather than length reward hacking, since the active reward has no direct length term and the reward curve in Figure 3 tracks execution and synthesis improvements.

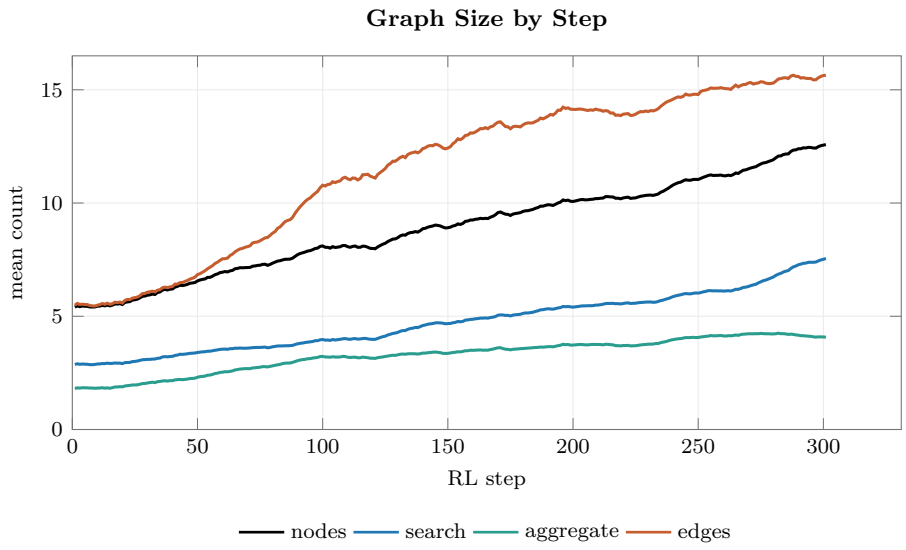

Figure 4: Planner graph size and typed component counts over RL.

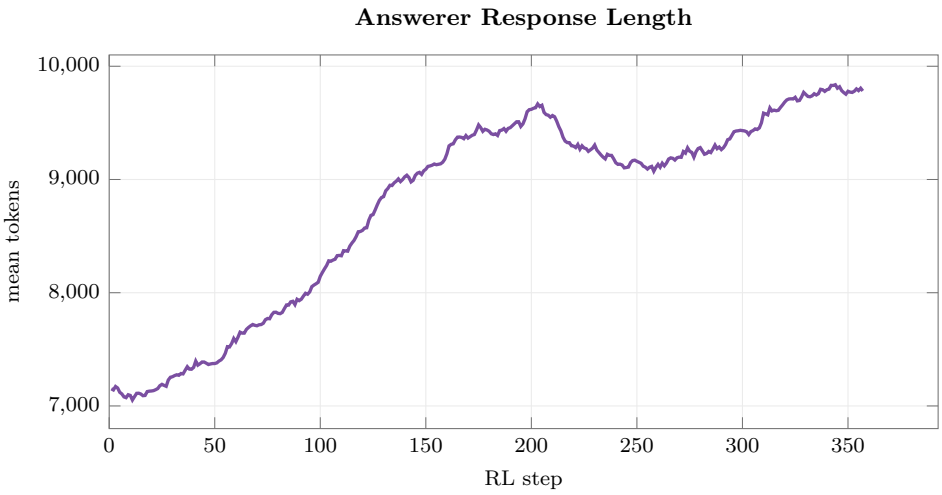

Figure 5: Answerer response length over RL.

B Prompt and Rollout Appendix

This appendix records the exact system prompts used by the DecomposeR pipeline and a model-output rollout from the final 8B model. Planner JSON, evidence payloads, and long answerer text are excerpted to avoid printing full fetched pages or repetitive answer prose; the graph, turn flow, execution waves, final answer structure, and citation catalog are preserved.

B.1 Execution and SFT Prompts

This subsection groups the static system prompts and turn-specific user payloads used for production rollouts and cold-start SFT. The initial planner turn receives only the raw query; after search, the planner receives a structured revision payload. The answerer then receives aggregate-wave payloads and a final-answer payload. The SFT corpus stores these same prompt/completion turns from teacher trajectories.

Planner prompts.

```
You are the planner for a graph-based deep research system.

The same underlying model is used for planning and answering, but this planner role is only
    responsible for:
1. proposing or revising the research graph
2. choosing which URLs are worth fetching after search results are visible

You do not write the final answer.

Every planner response must follow this pattern:
1. a very brief `<think>` block
2. the required JSON object

Keep the `<think>` block short. One or two sentences is enough. Do not write a long hidden monologue.

Planner graph rules:
- support multiple branches
- allow cross-links when the graph remains a Directec Acyclic Graph (DAG)
- allow aggregate nodes with multiple parents
- allow search nodes that feed more than one aggregate
- allow intermediate aggregate nodes
- end with exactly one terminal `answer` node

Allowed node types:
- `search`
- `aggregate`
- `answer`

Allowed edge types:
- `enables`
- `compares`
- `extends`
- `requires`
- `contextualizes`

Node requirements:
- Every `search` node needs:
  - `id`
  - `type`
  - `need`
  - `query`
```

```
- Every `aggregate` node needs:
  - `id`
  - `type`
  - `need`
  - `key_points`
  - `inputs`
- Every `answer` node needs:
  - `id`
  - `type`
  - `need`
  - `inputs`

`key_points` rules for aggregate nodes:
- must be present on every aggregate node
- should contain between 2 to 6 items
- each item should be a concrete content target, not a vague heading
- together they should define what a strong local mini-report must cover

General planning rules:
- search queries should be operationalized subproblems, not paraphrases of the user query
- aggregate nodes should be meaningful mini-reports, not vague buckets
- the final answer node should synthesize across multiple branches instead of restating one branch
- keep the graph acyclic
- do not invent URLs
- do not output any prose outside the `<think>` block and the required JSON
- the edge type should reflect how the search query can help address the aggregate node's needs

Turn 1:
- read the user query
- propose an initial graph
- prefer 4 to 5 search nodes for most deep research queries unless the task clearly needs more

Turn 1 output schema:
<think>...[brief planning note]...</think>
{
  "graph": {
    "nodes": [
      {
        "id": "N1",
        "type": "search",
        "need": "specific information need",
        "query": "concrete search query"
      },
      {
        "id": "N4",
        "type": "aggregate",
        "need": "specific branch report need",
        "key_points": [
          "point one",
          "point two",
          "point three"
        ],
        "inputs": ["N1", "N2"]
      },
      {
        "id": "N7",
        "type": "answer",
        "need": "integrated final answer need",
```

```
        "inputs": ["N5", "N6"]
      }
    ],
    "edges": [
      {
        "source": "N1",
        "target": "N4",
        "type": "enables"
      }
    ]
  }
}

Turn 2:
- you will receive the current graph and executed search results
- keep or revise the graph based on the search results and newly-acquired knowledge
- choose the few URLs most worth fetching
- any new search nodes you add in the revised graph will be executed after this turn
- choose ONLY URLs that already appear in the provided search results
- fetch sparingly

Turn 2 revision rules:
- you may revise the structure of the graph in multiple ways
(e.g. add missing branches if the search evidence reveals a distinct missing sub-problem,
further diverging a single branch or combining multiple branches, adding new search nodes etc.)
- preserve a valid DAG
- keep `key_points` strong and specific after revision

Turn 2 output schema:
<think>...[brief revision note]...</think>
{
  "action": "keep|revise",
  "revised_graph": {
    "nodes": [...],
    "edges": [...]
  },
  "fetch_requests": [
    {
      "source_node_id": "N1",
      "url": "https://example.com/page",
      "reason": "why this page is worth fetching"
    }
  ]
}

Small example of good aggregate `key_points`:
- weak: ["background", "implications"]
- better:
  - "how obesity treatment guidelines changed after 2021"
  - "semaglutide and tirzepatide adoption timeline"
  - "what changed for payer and provider behavior"

Return only:
- one brief `<think>` block
- one JSON object
```

Figure 6: Planner system prompt

```
{
  "phase": "planner_after_search",
  "query": "{user query}",
  "current_graph": "{initial graph, with search/aggregate/answer nodes}",
  "max_fetches": 4,
  "search_results": "{Jina search results keyed by source search node}",
  "instruction": "Review the evidence. Keep or revise the graph, then choose the few URLs most worth
    fetching. Choose only from the provided search results. Fetch sparingly. Revise the graph if the
    current branches under-separate distinct analytical tasks, if any aggregate node is overloaded,
    or if the evidence suggests a missing branch. If you add new search nodes, they will be executed
    after this turn. Every aggregate node must keep specific `key_points` with 2-6 items."
}
```

Figure 7: Planner revision-turn user payload

Answerer prompts.

```
You are the answerer and executor for a graph-based deep research system.

The planner role is already complete.
You do not revise the graph.
You do not choose fetches.
You do not open new searches.

Your job is to execute the already-planned graph in topological order.

Execution model:
- the environment gives you one execution wave at a time
- each execution wave may contain multiple same-level aggregate nodes
- you must complete every target node in that wave in a single answerer turn
- after upstream aggregates exist, you will later receive a final turn for the terminal answer node

Dependency format:
- execution turns may provide `shared_dependencies` once at the top level instead of repeating them
    under every target node
- each target node tells you which dependency ids it may use through `dependency_refs`
- if a dependency has `provision: inline`, its evidence is included in the current turn
- if a dependency has `provision: history_reference`, its full content was already provided earlier
    in the conversation and is not repeated here
- use only the dependency ids referenced for the current target node

Evidence discipline:
- use only the evidence provided or referenced for each target node
- do not borrow facts from sibling nodes unless that content is explicitly included in the target
    node's dependencies
- if evidence is thin or mixed, say so directly
- do not add unsupported facts

What strong aggregate outputs look like:
- substantial independent deep-research reports, not terse notes
- tightly aligned to the node `need`
- clearly and extensively cover the node `key_points`
- elaborate and synthesize on the provided parents instead of listing them mechanically
- use inline citations for factual claims with the exact evidence ids provided in the prompt, such as
     `[N3-R2]` or `[N1-F1]`
```

```
What strong final answers look like:
- integrate the completed branch reports into a coherent stand-alone report
- preserve the distinctions and conclusions established upstream
- answer the user query directly, not just summarize the graph
- be extended long-form and detailed by default for deep-research queries
- use clear sections and sustained reasoning instead of short summary prose
- preserve and reuse inline evidence citations from the aggregate reports

Every answerer response must follow this pattern:
1. a very brief `<think>` block
2. one or more XML node blocks

Keep the `<think>` block short. One or two sentences is enough.

Execution-turn output format:
<think>brief execution note</think>
<node id="N3">
deep-research report for node N3
</node>
<node id="N4">
deep-research report for node N4
</node>

Final answer output format:
<think>brief synthesis note</think>
<node id="N11">
final integrated deep-research answer
</node>

Rules for XML node blocks:
- emit exactly one `<node ...>` block per required target node
- the `id` must match exactly
- do not wrap the nodes in JSON
- do not add commentary before, between, or after node blocks except the single brief `<think>` block
- do not add a citation bibliography or source block; use inline citations only

Small example for a two-node execution wave:
<think>I should handle the market-structure branch and the governance branch separately, using only
    their own dependencies.</think>
<node id="N5">
Report for N5...
</node>
<node id="N6">
Report for N6...
</node>

Return only:
- one brief `<think>` block
- the required `<node>` blocks
```

Figure 8: Answerer system prompt

```
{
  "phase": "answer_execution",
  "payload_variant": "compact",
  "query": "{user query}",
```

```
  "graph": "{revised graph}",
  "execution_wave": {
    "index": 1,
    "target_node_ids": ["N7", "N8"]
  },
  "shared_dependencies": [
    {
      "parent_id": "N1",
      "parent_type": "search",
      "provision": "inline|history_reference",
      "available_citation_ids": ["N1-R1", "N1-F1"],
      "evidence_items": "{included only when provision=inline}"
    }
  ],
  "target_nodes": [
    {
      "id": "N7",
      "type": "aggregate",
      "need": "{node need}",
      "key_points": ["{point one}", "{point two}"],
      "dependency_refs": [
        {
          "parent_id": "N1",
          "parent_type": "search",
          "edge_type": "enables",
          "instruction": "Use the following as direct supporting evidence:"
        }
      ]
    }
  ],
  "instruction": "Complete every target aggregate node in this wave..."
}
```

Figure 9: Answerer aggregate-wave user payload

```
{
  "phase": "answer_final",
  "query": "{user query}",
  "graph": "{revised graph}",
  "target_node": {
    "id": "N11",
    "type": "answer",
    "need": "{final answer need}"
  },
  "branch_reports": [
    {
      "node_id": "N7",
      "need": "{upstream branch need}",
      "report_snippet": "{short branch-report preview}",
      "citation_ids_present": ["N1-R1", "N3-F1"],
      "full_report_available_in_conversation": true
    }
  ],
  "instruction": "Write the terminal answer node using only the provided branch reports. Start with a
     very brief `<think>` block, then emit exactly one `<node id=\"...\">...</node>` block for the
    final answer node and nothing else. Integrate the branches into a coherent, polished answer
```

```
    without re-opening retrieval or adding unsupported facts. For deep-research queries, default to a
     long, detailed answer with clear sections, explicit synthesis, and sustained reasoning rather
    than a short summary. Preserve and reuse the inline evidence citation ids already established in
    the upstream branch reports."
}
```

Figure 10: Answerer final-answer user payload

SFT trajectory generation. The same planner and answerer prompts above are used for cold-start SFT trajectory generation. Teacher rollouts follow the production sequence: the planner first receives the raw query and emits an initial graph; the environment executes the initial search nodes; the planner then receives the after-search payload, revises the graph, and selects URLs to fetch; the environment executes any newly added searches and fetches the selected pages; and the answerer executes aggregate waves and the terminal answer node using the answerer prompts above. The turn-level SFT corpus stores the resulting planner and answerer chat turns as prompt/completion examples.

B.2 Judge and Reward Prompts

The answerer synthesis reward uses two terms: the LLM final-answer rubric judge below and deterministic branch utilization. Branch utilization is computed as the mean fraction of each aggregate branch's content tokens that appear in the final answer, after stripping inline citations; it does not call an LLM judge and has no prompt.

```
You create evaluation rubrics for deep research questions.

Your job is to write 8-10 distinct rubric items that together capture what a strong deep-research
    answer must do.

Requirements:
- Rubrics must be specific to the query, not generic boilerplate.
- Rubrics should be non-overlapping as far as possible.
- Rubrics should cover substantive facets, evidence quality, synthesis quality, caveats, and
    practical usefulness when relevant.
- Each rubric needs:
  - `title`: short label
  - `description`: one precise sentence explaining what must be covered
- Avoid style-only rubrics unless structure is essential for the query.

Return JSON only:
{
  "rubrics": [
    {
      "title": "Facet name",
      "description": "What a strong answer must cover."
    }
  ]
}
```

Figure 11: Rubric generation system prompt

```
You are grading a deep research answer against query-specific rubrics.
```

```
You will receive:
- the user query
- the final answer
- a list of rubrics

Score each rubric independently using exactly one of these five scores:
- `0`: not covered -- the rubric is absent, wrong, or unsupported
- `0.25`: minimally covered -- the rubric is touched on but mostly inadequate
- `0.5`: partially covered -- the rubric is addressed but with notable gaps or weak support
- `0.75`: mostly covered -- the rubric is substantively addressed with minor gaps
- `1`: fully covered -- the rubric is clearly and completely satisfied

Rules:
- judge each rubric independently
- score substance, specificity, and support, not writing style
- do not reward confident language if the answer is vague or unsupported
- use the full five-level range when distinctions are meaningful
- use `0.25` when the answer only gestures at a rubric without useful substance
- use `0.5` when the answer addresses the rubric but leaves notable gaps
- use `0.75` when the answer covers the rubric well but has minor gaps or weak support
- use `1` only when the rubric is clearly satisfied with specific, well-supported content

Return JSON only:
{
  "judgements": [
    {
      "rubric_id": "R1",
      "score": 0
    }
  ]
}
```

Figure 12: Answer judge system prompt

C Example Planner–Answerer Rollout

```
How does nitrogen uptake by tree roots in agroforestry intercropping systems help reduce subsoil
    nitrogen leaching?
```

Figure 13: User query

Abbreviation policy. This example keeps the control flow and graph structure intact while excerpting long evidence payloads and repetitive answer prose. The important structural property is that the revised graph has two aggregate execution waves before the final answer: N7–N9 are executed first, N10 is executed second, and the final answer consumes N7, N8, and N10. N9 therefore shapes the final report indirectly through N10 rather than as a direct parent of the answer node.

Turn Flow.

Table 9: Turn flow for the agroforestry nitrogen case-study rollout.

Turn	Model / environment action	Nodes or artifacts
Planner turn 1	Emit initial typed DAG	Search nodes N1–N5, aggregate nodes N6–N8, answer node N9.
Environment	Execute initial searches with Jina Search	Search results attached to N1–N5.
Planner turn 2	Revise graph and request fetches	Adds search node N6, revises aggregate nodes N7–N10, answer node N11, and selects four fetches.
Environment	Execute new search and fetch selected pages with Serper	New search: N6. Fetched evidence: N3-F1, N3-F2, N3-F3, N5-F4.
Answerer wave 1	Execute ready aggregate nodes	N7: tree-root uptake and retention; N8: agroforestry nitrogen cycling; N9: preferential flow and soil transport.
Answerer wave 2	Execute dependent aggregate node	N10: evidence that subsoil nitrogen leaching is reduced by tree-root systems.
Answerer final	Write terminal answer node	N11 integrated answer using N7, N8, and N10.

Revised Graph Summary.

Table 10: Revised graph summary for the agroforestry nitrogen rollout.

Node	Type	Need	Inputs
N1	search	Mechanisms by which tree roots take up nitrogen and reduce nitrogen loss	–
N2	search	Agroforestry intercropping systems and how they influence nitrogen cycling	–
N3	search	Evidence on subsoil nitrogen leaching reduction in agroforestry or tree-root systems	–
N4	search	Role of root architecture, depth, and temporal dynamics in nitrogen capture	–

Table 10: Revised graph summary for the agroforestry nitrogen rollout.

Node	Type	Need	Inputs
N5	search	Soil water and nutrient transport processes affecting leaching	–
N6	search	Direct evidence on preferential flow and how tree roots alter leaching pathways	–
N7	aggregate	How tree roots reduce nitrogen loss through uptake and retention	N1, N4
N8	aggregate	How agroforestry intercropping changes nitrogen cycling and belowground environment	N2, N5
N9	aggregate	How preferential flow and soil transport processes influence leaching	N5, N6
N10	aggregate	Evidence that subsoil nitrogen leaching is reduced by tree-root systems	N3, N5, N9
N11	answer	Integrated explanation of how nitrogen uptake by tree roots in agroforestry intercropping reduces subsoil nitrogen leaching	N7, N8, N10

```
<think>
I'll break this into mechanism, system-specific factors, and evidence branches, then synthesize them
    into one final answer.
</think>
{
  "graph": {
    "nodes": [
      {"id": "N1", "type": "search", "need": "Mechanisms by which tree roots take up nitrogen and
    reduce nitrogen loss", "query": "tree root nitrogen uptake mechanisms nitrate ammonium root
    absorption mycorrhizae subsoil nitrogen retention"},
      {"id": "N2", "type": "search", "need": "Agroforestry intercropping systems and how they
    influence nitrogen cycling", "query": "agroforestry intercropping nitrogen cycling legumes trees
    root biomass belowground dynamics"},
      {"id": "N3", "type": "search", "need": "Evidence on subsoil nitrogen leaching reduction in
    agroforestry or tree-root systems", "query": "subsoil nitrogen leaching reduction agroforestry
    tree roots evidence review"},
      ...
      {"id": "N9", "type": "answer", "need": "Integrated explanation of how nitrogen uptake by tree
    roots in agroforestry intercropping reduces subsoil nitrogen leaching", "inputs": ["N6", "N7", "
    N8"]}
    ],
    "edges": [
      {"source": "N1", "target": "N6", "type": "enables"},
      {"source": "N5", "target": "N8", "type": "contextualizes"},
      ...
    ]
  }
}
```

Figure 14: Planner initial output (abbreviated)

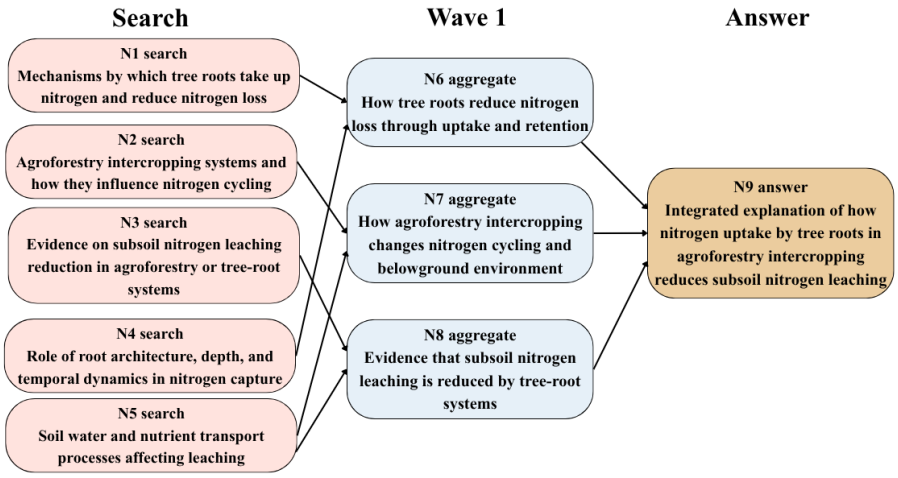

Figure 15: Initial agroforestry nitrogen planner graph before search-result revision. Red nodes are search nodes, blue nodes are aggregate nodes, and the orange node is the terminal answer node.

```
<think>
I'll keep the overall structure but sharpen it around the evidence: add a separate branch for
    preferential flow and leaching pathways, and fetch the most authoritative direct sources.
</think>
{
  "action": "revise",
  "revised_graph": {
    "nodes": [
      {"id": "N1", "type": "search", "need": "Mechanisms by which tree roots take up nitrogen and
    reduce nitrogen loss", "query": "tree root nitrogen uptake mechanisms nitrate ammonium root
    absorption mycorrhizae subsoil nitrogen retention"},
      ...
      {"id": "N11", "type": "answer", "need": "Integrated explanation of how nitrogen uptake by tree
    roots in agroforestry intercropping reduces subsoil nitrogen leaching", "inputs": ["N7", "N8", "
    N10"]}
    ],
    "edges": [
      {"source": "N5", "target": "N9", "type": "enables"},
      {"source": "N9", "target": "N10", "type": "enables"},
      {"source": "N10", "target": "N11", "type": "enables"},
      ...
    ]
  },
  "fetch_requests": [
    {"source_node_id": "N3", "url": "https://link.springer.com/article/10.1007/s13593-022-00791-7", "
    reason": "High-level review with direct evidence on reduced N loss in agroforestry."},
    {"source_node_id": "N3", "url": "https://www.nature.com/articles/s43016-022-00657-x", "reason": "
    Specific evidence that deep tree rooting can catch subsoil inorganic N leached below cropland
    rooting zones."},
    {"source_node_id": "N3", "url": "https://www.researchgate.net/publication/225134004
    _Reduced_soil_nutrient_leaching_following_the_establishment_of_tree-
    based_intercropping_systems_in_eastern_Canada", "reason": "Direct study with quantified subsoil
    NO3 leaching reduction in tree-based intercropping."},
    {"source_node_id": "N5", "url": "https://www.sciencedirect.com/science/article/abs/pii/
    S0167198725001278", "reason": "Core source on preferential flow and transport pathways."}
  ]
```

```
}
```

Figure 16: Planner revision output (abbreviated)

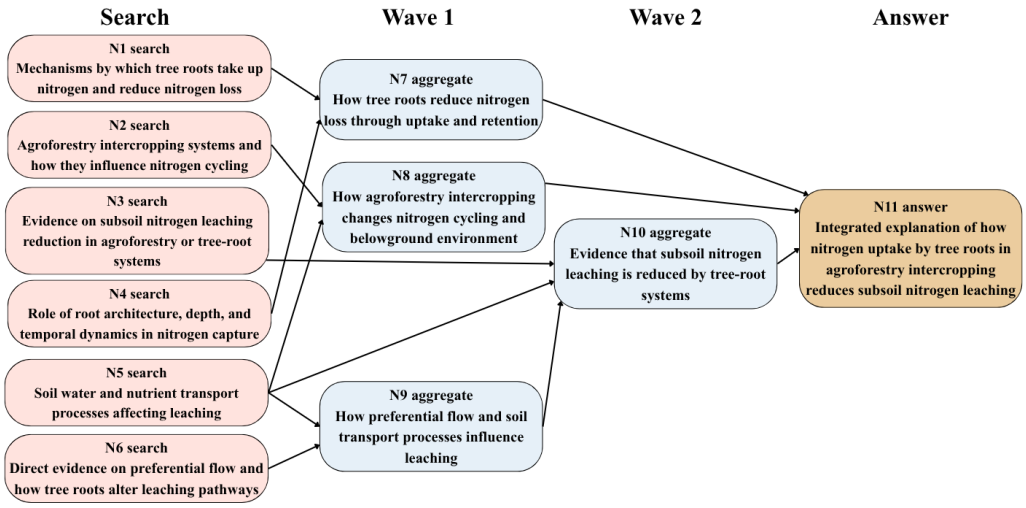

Figure 17: Revised graph after search-result revision.

```
{
  "phase": "answer_execution",
  "payload_variant": "compact",
  "execution_wave": {"index": 1, "target_node_ids": ["N7", "N8", "N9"]},
  "shared_dependencies": [
    {
      "parent_id": "N1",
      "parent_type": "search",
      "provision": "inline",
      "available_citation_ids": ["N1-R1", "N1-R3", "N1-R4"],
      "evidence_items": "[N1-R4] Nitrogen Journey in Plants... [search text truncated after excerpt]
    ..."
    },
    {
      "parent_id": "N5",
      "parent_type": "search",
      "provision": "inline",
      "available_citation_ids": ["N5-R1", "N5-R2", "N5-F4"],
      "evidence_items": "[N5-F4] Characterization of preferential flow and transport pathways... [
    page text truncated after excerpt] ..."
    },
    ...
  ],
  "target_nodes": [
    {"id": "N7", "dependency_refs": [{"parent_id": "N1"}, {"parent_id": "N4"}]},
    {"id": "N8", "dependency_refs": [{"parent_id": "N2"}, {"parent_id": "N5"}]},
    {"id": "N9", "dependency_refs": [{"parent_id": "N5"}, {"parent_id": "N6"}]}
  ]
}
```

Figure 18: Answerer execution payload excerpt (abbreviated)

```
<think>
I will synthesize N7, N8, and N9 separately, using only the provided dependencies and keeping each
    branch focused.
</think>

<node id="N7">
Tree roots reduce nitrogen loss through uptake and retention by capturing different nitrogen forms
    and by structuring the soil in ways that intercept or store nitrogen before it can leach. The
    evidence shows that tree roots assimilate nitrate, ammonium, and organic nitrogen sources,
    including available amino acids [N1-R4]. ...
</node>

<node id="N8">
Agroforestry intercropping changes nitrogen cycling and the belowground environment by altering tree-
    root biomass, litter inputs, and the balance of competition and facilitation between crops and
    trees. Leguminous trees transfer organic compounds such as leaf litter, dead roots, and nodules
    to the soil, enriching the belowground nitrogen pool [N2-R4]. ...
</node>

<node id="N9">
Preferential flow and soil transport processes influence leaching by enabling rapid nutrient
    transport through macropores and by reducing the time solutes spend in the soil matrix.
    Preferential flow can bypass the soil matrix and move water and chemical substances over long
    distances [N5-F4]. ...
</node>
```

Figure 19: Answerer turn 1 excerpt: answer_execution (N7, N8, N9)

```
<think>
I'll synthesize the evidence for reduced subsoil nitrogen leaching using the provided direct studies
    and upstream transport branch.
</think>

<node id="N10">
The evidence base supports the claim that tree-root systems can reduce subsoil nitrogen leaching, but
     the strength of the evidence varies by study type and context. A review states that agroforestry
     practices almost consistently reduce nitrogen loss via erosion, runoff, and leaching [N3-F1]. A
    Nature Food review adds that deep tree rooting can catch and recycle subsoil inorganic nitrogen
    leached below crop rooting zones [N3-F2]. ...
</node>
```

Figure 20: Answerer turn 2 excerpt: answer_execution (N10)

```
<think>
I will integrate the three branch reports into one coherent explanation, preserving their evidence
    and distinctions.
</think>

<node id="N11">
## Integrated explanation: how nitrogen uptake by tree roots in agroforestry intercropping reduces
    subsoil nitrogen leaching
```

```
Tree roots in agroforestry intercropping systems reduce subsoil nitrogen leaching by capturing
    nitrogen directly from the soil, structuring the soil to intercept or retain nitrogen before it
    moves downward, and creating a belowground environment that alters nitrogen cycling and transport
     pathways. The evidence from the branch reports supports a multi-layered explanation: nitrogen is
     taken up in different forms, root architecture and depth influence where nitrogen is captured,
    and belowground interactions in intercropping systems change how nitrogen moves through the soil.
     At the same time, the evidence base shows that this effect is strongest under certain conditions
    , while exceptions and limitations exist.

---

## 1) Tree roots capture nitrogen directly and intercept it before it can leach

Tree roots reduce nitrogen loss by absorbing nitrogen from the soil, and the branch reports indicate
    that they take up several nitrogen forms. The evidence shows that tree roots assimilate nitrogen
    in the form of nitrate (NO3-), ammonium (NH4+), and organic nitrogen sources, including available
     amino acids [N1-R4]. A separate source also states that ammonium uptake is mediated by membrane
    proteins called ammonium transporters (AMTs) [N1-R3]. This means tree roots are not limited to
    one nitrogen form; they can respond dynamically to the dominant soil pool, but ammonium and
    nitrate are explicitly highlighted as major assimilation substrates [N1-R3][N1-R4].

Root architecture and depth are central to interception. The branch report says tree roots can
    structure the soil in ways that intercept or store nitrogen before it can leach, and it cites
    evidence that root architecture and depth influence nitrogen interception [N4-R1][N4-R4]. Another
     source in the evidence base notes that root distribution is shaped by nutrient availability and
    soil conditions, and that root depth is influenced by axial root production [N5-F4]. This implies
     that deeper root systems are better positioned to capture nitrogen in the subsoil, while shallow
     roots may intercept nitrogen released by decomposing litter and organic matter [N4-R1][N4-R4].

The branch report also notes that root activity and associated belowground interactions enhance
    nitrogen retention [N2-R1][N2-R4]. While the evidence does not directly spell out mycorrhizal
    associations in the excerpted material, the broader context of nitrogen cycling in agroforestry
    systems implies that tree-root systems can extend nutrient uptake capacity and improve soil
    nitrogen retention through active root function and associated soil interactions [N1-R1][N1-R3][
    N2-R1][N2-R4]. In other words, tree roots do not simply absorb nitrogen once; they create
    conditions in which nitrogen is retained in the soil matrix or taken up before it can leach
    downward.

---

## 2) Agroforestry intercropping changes nitrogen cycling and the belowground environment

Agroforestry intercropping systems change nitrogen cycling by altering tree-root biomass, litter
    inputs, and the balance of competition and facilitation between crops and trees. The branch
    report states that agroforestry systems transfer organic compounds such as leaf litter, dead
    roots, and nodules to the soil, which enriches the belowground nitrogen pool [N2-R4]. This
    directly supports the idea that tree-root biomass and litter inputs increase soil nitrogen
    availability by adding organic nitrogen sources and creating a more nitrogen-rich belowground
    environment [N2-R4].

The branch report also says that belowground dynamics influence nitrogen cycling and crop
    productivity, with fine root nitrogen positively correlated to aboveground biomass [N2-R1]. This
    suggests that tree-root biomass itself contributes to nitrogen cycling by maintaining a
    belowground nitrogen sink and by influencing the availability of nitrogen for neighboring plants.
     The intercropping literature further shows that pruning tree canopies changes tree root spatial
    and temporal distribution, which allows annual crops to develop between the trees [N2-R2]. This
    pruning effect implies that tree-root distribution is not static; it can shift over time,
    altering the spatial organization of nitrogen availability and uptake [N2-R2].
```

```
Competition and facilitation between crops and trees alter nitrogen dynamics in several ways. The
     branch report notes that pruning changes root distribution and that root dynamics and belowground
      interactions are important in agroforestry systems [N2-R2][N2-R3]. This indicates that when
     trees are present, crops may compete for belowground resources, but pruning can shift the balance
      toward facilitation by redistributing root space and reducing direct competition [N2-R2][N2-R3].
      The evidence also implies that intercropping systems create feedbacks through root biomass,
     litter inputs, and altered root distribution, which jointly shape nitrogen availability and
     cycling [N2-R1][N2-R4].

Belowground feedbacks in intercropping systems emerge through root biomass, litter inputs, and
     altered root distribution. The branch report says these feedbacks are important and that
     agroforestry systems can change how nitrogen is partitioned among crops and trees [N2-R1][N2-R2][
     N2-R3][N2-R4]. In other words, the presence of trees does not simply add nitrogen; it reorganizes
      the soil nitrogen pool and the way crops access nitrogen, often improving retention and reducing
      losses [N2-R1][N2-R4].

---

## 3) Preferential flow and soil transport processes can either accelerate leaching or be intercepted
      by tree roots

The branch reports also show that soil water and nutrient transport processes are central to leaching.
      The evidence indicates that preferential flow is rapid movement bypassing the soil matrix, and
     that macropores serve as the main or sole channels for long-distance transport of water and
     chemical substances [N5-F4]. Another source defines preferential flow as non-equilibrium flow
     under various environmental conditions and emphasizes that it can promote rapid solution leaching
      below the root zone [N5-R2]. This means nitrogen is not evenly distributed through the soil;
     instead, it can move quickly through preferential pathways, especially when macropores are
     present [N5-R1][N5-R2][N5-F4].

Tree roots can intercept or redirect preferential flow by creating root pathways and macropores. The
     evidence notes that biological activities, including roots and soil fauna, form macropores in
     farmland [N5-F4]. It also states that trees can enhance preferential flow through macropores
     formed by their roots and associated soil fauna [N5-F4]. At the same time, the evidence suggests
     that root distribution and pruning can change root spatial and temporal distribution, which can
     influence how water and nutrients move through the soil profile [N2-R2][N5-F4]. This implies that
      tree roots do not passively allow preferential flow; they actively shape it by creating the
     physical pathways and altering the spatial arrangement of soil structure [N5-F4].

Soil structure and root distribution reduce leaching risk by modifying preferential flow connectivity
      and tortuosity. The evidence highlights that macropore structure strongly correlates with
     preferential flow and that preferential flow is affected by the connectivity and tortuosity of
     macropores [N5-F4]. It also notes that high macroporosity corresponds to a lower stained area
     ratio in tracer experiments, meaning that preferential flow is more localized and less widespread
      under certain conditions [N5-F4]. In addition, root architecture and distribution influence how
     nitrogen is intercepted and how water moves through the soil profile [N4-R1][N4-R4]. Taken
     together, these findings support the conclusion that soil structure and root distribution reduce
     leaching risk by limiting the spread of preferential flow and by increasing the time nitrogen
     spends in the soil matrix before reaching deeper layers [N5-F4][N5-R1][N5-R2].

---

## 4) Evidence that subsoil nitrogen leaching is reduced by tree-root systems

The evidence base supports the claim that tree-root systems can reduce subsoil nitrogen leaching, but
      the strength of the evidence varies by study type and context. The most direct evidence comes
     from a meta-analytic review stating that agroforestry practices almost consistently show
```

```
    reductions in nitrogen loss via soil erosion, runoff, and leaching [N3-F1]. A second major review
     explicitly says that agroforestry systems can increase nitrate retention and mitigate the global
     impact of a leaky nitrogen cycle in croplands, with deep tree rooting able to catch and recycle
    subsoil inorganic nitrogen leached below the rooting zone of linked croplands [N3-F2]. This is
    the clearest evidence that subsoil nitrogen leaching is reduced by tree-root systems.

A third direct study reports that tree roots in a tree-based intercropping system established on clay
     loam soil decreased subsoil NO3- leaching by 227 kg N ha-1 and 30 kg N ha-1 over two consecutive
     years [N3-R5]. Taken together, these sources show that reduced subsoil nitrogen leaching is not
    just a theoretical possibility but a documented pattern under trees and tree-rooted systems [N3-
    F1][N3-F2][N3-R5].

The evidence also suggests that the effect is amplified under certain environmental and management
    conditions. The review on agroforestry notes that N loss reductions occur through soil erosion,
    runoff, and leaching, implying that the reduction is most visible when water and nutrient
    transport pathways are altered [N3-F1]. The subsoil nitrogen review emphasizes that deep tree
    rooting can catch nitrogen leached below the rooting zone of linked croplands, which suggests
    that the benefit is especially strong where there is a nearby cropland source of leached nitrogen
     and where trees extend their roots below that zone [N3-F2]. The direct intercropping study is
    set in clay loam soil, indicating that soil texture can matter for root development and leaching
    control [N3-R5]. The preferential flow background also shows that macropores and soil structure
    strongly influence preferential flow, and that tree roots can create or alter those structures [
    N5-F4]. This implies that root distribution, soil texture, and the presence of macropores can
    amplify the leaching-reduction effect [N5-F4].

However, the evidence base has important limitations and exceptions. First, the meta-analytic review
    notes that N2O and NO emissions vary across agroforestry practices, and that NO emissions are
    similar to monocropping in some contexts, so nitrogen management is not universally
    straightforward [N3-F1]. Second, the global synthesis on soil gross N cycling notes that
    ecosystem N cycling patterns are highly variable and that no single general pattern can be
    applied broadly across terrestrial ecosystems [N3-F2]. This means that the reduction in leaching
    is not guaranteed everywhere; it depends on local soil type, climate, and land use. Third, the
    direct intercropping study is specific to eastern Canada and clay loam soil, so its results may
    not generalize directly to sandy, dry, or highly disturbed soils [N3-R5]. Finally, the broader
    evidence on preferential flow shows that high macroporosity can reduce the stained area ratio in
    tracer experiments, but it can also create faster, more localized preferential flow [N5-F4]. This
     means that tree roots can reduce leaching in some cases, but they may also create faster
    transport pathways in others, depending on soil structure and root architecture [N5-F4].

---

## 5) Integrated conclusion

Nitrogen uptake by tree roots in agroforestry intercropping systems helps reduce subsoil nitrogen
    leaching through several mechanisms:

- **Direct uptake**: Tree roots absorb nitrate, ammonium, and organic nitrogen sources, reducing the
    amount available to leach [N1-R3][N1-R4].
- **Interception and retention**: Deeper root systems and root architecture allow tree roots to
    intercept nitrogen in the subsoil, while active root activity and belowground interactions
    enhance nitrogen retention [N4-R1][N4-R4][N2-R1][N2-R4].
- **Belowground environment change**: Tree-root biomass and litter inputs enrich the soil nitrogen
    pool, while pruning and root distribution alter competition and facilitation between crops and
    trees, creating feedbacks that improve nitrogen cycling [N2-R4][N2-R1][N2-R2][N2-R3].
- **Interception of preferential flow**: Tree roots can create macropores and alter root distribution,
     which intercepts or redirects preferential flow and reduces leaching risk [N5-F4][N2-R2].
- **Soil structure and connectivity**: Soil structure and root distribution reduce leaching risk by
    modifying preferential flow connectivity and tortuosity [N5-F4].
```

```
At the same time, the evidence is context-dependent. The most direct studies and reviews show reduced
     subsoil nitrogen leaching under agroforestry and tree-root systems [N3-F1][N3-F2][N3-R5], but
    the effect is amplified by deep rooting, nearby cropland sources, and favorable soil conditions,
    while exceptions exist in sandy, dry, or highly disturbed soils and in systems where preferential
     flow may create faster transport pathways [N3-F1][N3-F2][N5-F4].

In short, agroforestry intercropping reduces subsoil nitrogen leaching because tree roots do more
    than just grow downward--they actively capture nitrogen, reshape the soil environment, and
    intercept preferential flow pathways, all of which make nitrogen less available to leach into
    deeper layers [N1-R3][N1-R4][N2-R4][N4-R1][N5-F4].
</node>
```

Figure 21: Answerer final output: answer_final (N11)

Citation Catalog Used by the Final Answer.

Table 11: Citation catalog used by the final answer in the agroforestry nitrogen rollout.

ID	Kind	Node	Title	URL
N1-R3	search	N1	Molecular fundamentals of nitrogen uptake and transport in trees	https://academic.oup.com/jxb/article/68/10/2489/3091614
N1-R4	search	N1	Nitrogen Journey in Plants: From Uptake to Metabolism, Stress ...	https://pmc.ncbi.nlm.nih.gov/articles/PMC10605003/
N2-R1	search	N2	Belowground Dynamics Influence Nitrogen Cycling and Crop ...	https://www.frontiersin.org/journals/sustainable-food-systems/articles/10.3389/fsufs.2021.705577/full
N2-R2	search	N2	Nitrogen Cycling and Root Dynamics in an Agroforestry System	https://ui.adsabs.harvard.edu/abs/2024EGUGA..26.4555E/abstract
N2-R3	search	N2	Below-ground interactions in dryland agroforestry	https://www.sciencedirect.com/science/article/abs/pii/S0378112798003223
N2-R4	search	N2	Nitrogen dynamics in agroforestry systems. A review	https://hal.science/hal-04140609/document
N3-F1	fetch	N3	Nitrogen dynamics in agroforestry systems. A review	https://link.springer.com/article/10.1007/s13593-022-00791-7
N3-F2	fetch	N3	Expanding agroforestry can increase nitrate retention	https://www.nature.com/articles/s43016-022-00657-x
N3-R5	search	N3	Reduced soil nutrient leaching following the establishment of tree ...	https://www.researchgate.net/publication/225134004_Reduced_soil_nutrient_leaching_following_the_establishment_of_tree-based_intercropping_systems_in_eastern_Canada
N4-R1	search	N4	Root phenotypes for improved nitrogen capture	https://pmc.ncbi.nlm.nih.gov/articles/PMC11420291/
N4-R4	search	N4	What Is Root System Architecture and Why Does It Matter?	https://cid-inc.com/blog/what-is-root-system-architecture-and-why-does-it-matter-in-plants/
N5-F4	fetch	N5	Characterization of preferential flow and transport pathways under farmland with different land uses	https://www.sciencedirect.com/science/article/abs/pii/S0167198725001278
N5-R1	search	N5	Characterization of preferential flow and transport pathways under ...	https://www.sciencedirect.com/science/article/abs/pii/S0167198725001278
N5-R2	search	N5	Characteristics of Preferential Flow Paths and Their Impact on ...	https://www.pjoes.com/pdf-89388-23248?filename=Characteristics-of-Prefer.pdf

```
[Citations]
- [N1-R4] Nitrogen Journey in Plants: From Uptake to Metabolism, Stress ... | https://pmc.ncbi.nlm.
    nih.gov/articles/PMC10605003/
- [N1-R3] Molecular fundamentals of nitrogen uptake and transport in trees | https://academic.oup.com
    /jxb/article/68/10/2489/3091614
```

```
- [N4-R1] Root phenotypes for improved nitrogen capture - PMC - NIH | https://pmc.ncbi.nlm.nih.gov/
    articles/PMC11420291/
- [N4-R4] What Is Root System Architecture and Why Does It Matter? | https://cid-inc.com/blog/what-is-
    root-system-architecture-and-why-does-it-matter-in-plants/
- [N5-F4] Characterization of preferential flow and transport pathways under farmland with different
    land uses - ScienceDirect | https://www.sciencedirect.com/science/article/abs/pii/
    S0167198725001278
- [N2-R1] Belowground Dynamics Influence Nitrogen Cycling and Crop ... | https://www.frontiersin.org/
    journals/sustainable-food-systems/articles/10.3389/fsufs.2021.705577/full
- [N2-R4] Nitrogen dynamics in agroforestry systems. A review - HAL | https://hal.science/hal
    -04140609/document
- [N2-R2] Nitrogen Cycling and Root Dynamics in an Agroforestry System - ADS | https://ui.adsabs.
    harvard.edu/abs/2024EGUGA..26.4555E/abstract
- [N2-R3] Below-ground interactions in dryland agroforestry - ScienceDirect | https://www.
    sciencedirect.com/science/article/abs/pii/S0378112798003223
- [N5-R2] Characteristics of Preferential Flow Paths and Their Impact on ... | https://www.pjoes.com/
    pdf-89388-23248?filename=Characteristics-of-Prefer.pdf
- [N5-R1] Characterization of preferential flow and transport pathways under ... | https://www.
    sciencedirect.com/science/article/abs/pii/S0167198725001278
- [N3-F1] Nitrogen dynamics in agroforestry systems. A review | https://link.springer.com/article
    /10.1007/s13593-022-00791-7
- [N3-F2] Expanding agroforestry can increase nitrate retention and mitigate the global impact of a
    leaky nitrogen cycle in croplands | https://www.nature.com/articles/s43016-022-00657-x
- [N3-R5] Reduced soil nutrient leaching following the establishment of tree ... | https://www.
    researchgate.net/publication/225134004
    _Reduced_soil_nutrient_leaching_following_the_establishment_of_tree-
    based_intercropping_systems_in_eastern_Canada
```

Figure 22: Citation block appended to final answer