SearchOS-V1: Towards Robust Open-Domain Information-Seeking Agent Collaboration

Yuyao Zhang[1,2,*,‡], **Junjie Gao**[2,*], **Zhengxian Wu**[2], **Jiaming Fan**[2], **Jin Zhang**[2], **Shihan Ma**[2], **Yao Yao**[2], **Weiran Qi**[2], **Chuyan Jin**[2], **Guiyu Ma**[2], **Xingzhong Xu**[2], **Kai Yang**[2], **Ji-Rong Wen**[1], **Zhicheng Dou**[1,†]

[1]Gaoling School of Artificial Intelligence, Renmin University of China [2]Ant Group

[*]Equal contribution, [†]Corresponding author, [‡]Work done during an internship at Ant Group

Abstract

Recent advances in Tool-Integrated Large Language Models have made web search a core capability of information-seeking agents. However, as interaction histories grow, agents increasingly struggle to track task progress. When search attempts fail to yield useful evidence, current single- and multi-agent systems can become trapped in repetitive loops, wasting search budgets and ultimately compromising the quality and completeness of the final output. We introduce **SearchOS**, a system-level multi-agent framework that turns fragile, implicit search progress into explicit, persistent, and shared state. First, we formulate open-domain information seeking as *relational schema completion with grounded citations*, where agents discover entities, populate attributes across linked tables, and anchor each value to source evidence. Then we design Search-Oriented Context Management (SOCM), which externalizes the evolving state into Frontier Task, an Evidence Graph, a Coverage Map, and Failure Memory. Built on SOCM, **SearchOS** applies a pipeline-parallel scheduling mechanism that overlaps the execution of sub-agents and continuously refills freed slots with tasks targeting unresolved coverage gaps to improve utilization and throughput. To schedule and control the execution of search agents, **SearchOS** introduces a Search Tool Middleware Harness that intercepts model and tool interactions to record grounded evidence and react to stalls or budget exhaustion, and provides a reusable hierarchical skill system comprising strategy and access skills to augment the agents' search process and avoid repeating failed search patterns across runs. On WideSearch and GISA, **SearchOS** leads all metrics among the evaluated single- and multi-agent baselines, paving the way toward robust information-seeking collaboration.

Website: https://antins-labs.github.io/SearchOS
Code: https://github.com/antins-labs/SearchOS
YouTube: https://www.youtube.com/playlist?list=PLXMUs3Ayz3EQ

GAOLING SCHOOL OF ARTIFICIAL INTELLIGENCE

ANT GROUP

1 Introduction

Recent advances in tool-integrated large language models (LLMs) have made web search a core capability of information-seeking agents (Yao et al., 2022; Nakano et al., 2021; Jin et al., 2025a; Li et al., 2026b). These agents can iteratively search, browse, and reason over external sources, extending language models beyond the knowledge available in their parameters. As the field moves toward broad, open-domain and long-horizon tasks (Wong et al., 2025; Zhu et al., 2026), an agent must gather large collections of facts, reconcile evidence across sources, and produce a structured

Figure 1 **SearchOS** interface for a long-horizon information-seeking task. The workspace exposes the orchestration trace, pipeline parallel agent activity, and relational schema coverage.

answer with verifiable citations.

Stronger search capability alone, however, does not make this process reliable. As interaction histories grow, agents lose track of what has been established and what remains unresolved. Evidence becomes buried in context, making omissions, redundant collection, and conflicting claims difficult to detect. When a search path yields no useful information, a single agent may repeatedly issue similar searches or continue exploring the same dead end, wasting its budget and degrading the final answer. Simply adding more agents does not eliminate these problems: parallel workers may duplicate effort, disagree on the intended fields, or leave execution slots idle while the system waits for the slowest task.

These failures arise because conventional agents treat plans, progress, evidence, and failures as transient conversation content. **Search state should be maintained by the system rather than inferred repeatedly from interaction history**. A long-horizon search system should expose what entities and attributes have been covered, preserve the provenance of every value, coordinate workers around explicit gaps, and intervene when execution stalls. It should also retain effective search and access patterns across sessions instead of rediscovering them for every task.

Based on this principle, we present **SearchOS**, a multi-agent framework for structured and observable open-domain information seeking. We first formulate an information-seeking task as *relational schema completion with grounded citations*. Following relational modeling and normalization principles (Codd, 1970; Bernstein, 1976), a relational search schema may contain multiple tables connected through primary–foreign-key relations. The system jointly discovers the entity rows, fills their attributes, and maintains a citation matrix that maps each value to a source URL and an anchored excerpt. This formulation, related to recent table-completion views of long-horizon information seeking (Lan et al., 2026), turns an open-ended request into a concrete search target and makes progress measurable at the level of entities and attributes.

We then introduce **Search-Oriented Context Management** (SOCM) to externalize execution state and share Frontier Task, an Evidence Graph, a Coverage Map, and Failure Memory across all agents. A central orchestrator decomposes the schema into coverage-oriented tasks and coordinates specialized explore, search, and writer agents. Rather than dispatching workers in synchronized batches, **SearchOS** uses pipeline-parallel scheduling: different sub-tasks progress concurrently, and each released agent slot is immediately assigned a new unresolved schema gap. This design follows the utilization advantage demonstrated by pipeline parallelism in GPU training (Huang et al., 2019; Narayanan et al., 2019); it reduces straggler-induced idle time and increases search throughput.

Reliable execution also requires controls that should not depend on an agent remembering to invoke them. We therefore introduce a **Search Tool Middleware Harness** that intercepts model and tool interactions to inject relevant state, extract and anchor evidence, enforce budgets, and detect repeated or stalled trajectories. Agents can focus on local search decisions while the harness maintains global execution invariants. Finally, we design a hierarchical search skill system that separates reusable strategy skills—how to search—from site-specific access skills—how to reach and extract information from a source. The system routes these skills by task and source and refines them from successful and failed trajectories.

Experiments on WideSearch and GISA show that **SearchOS** leads all reported F1 metrics among the evaluated single- and multi-agent baselines. It achieves 80.3 item-level F1 on WideSearch and 76.5 set F1 on GISA, exceeding the strongest baseline on the latter by 13.4 points, paving the way toward robust information-seeking collaboration. Our main contributions are as follows:

1. **Relational Search Formulation.** We formulate open-domain information seeking as relational schema completion with grounded citations, providing a unified and verifiable objective for entity discovery, attribute completion, and evidence attribution.
2. **Stateful, Pipeline-Parallel Collaboration.** We introduce SOCM and a continuous orchestration scheme that coordinate specialized agents through shared search state and dynamically assign work around unresolved coverage gaps.
3. **Middleware-Governed Search Agent Execution.** We introduce a Search Tool Middleware Harness that moves evidence processing, context management, budget enforcement, and behavioral safeguards outside agent prompts.
4. **Hierarchical Search Skills.** We design a hierarchical search agent skills system, comprising reusable search strategies and site-specific access procedures across sessions to augment the agents' search process.

2 Problem Formulation

We formulate open-domain information seeking as **relational schema completion with grounded citations**. Following relational database design, we use primary keys to identify entities, foreign keys to express cross-table relations, and normalized tables to reduce redundant or inconsistent fact representations (Codd, 1970; Bernstein, 1976).

Given a natural-language request q, the system constructs a relational search schema

$$\mathcal{S} = \big(\{T_m\}_{m=1}^{M}, \mathcal{R}\big), \qquad T_m = \big(\mathcal{A}_m, \mathcal{P}_m\big), \tag{1}$$

where T_m defines a table with attributes $\mathcal{A}_m$ and primary-key attributes $\mathcal{P}_m$, and $\mathcal{R}$ contains foreign-key relations between tables. The task is denoted by $\mathcal{T} = (q, \mathcal{S})$.

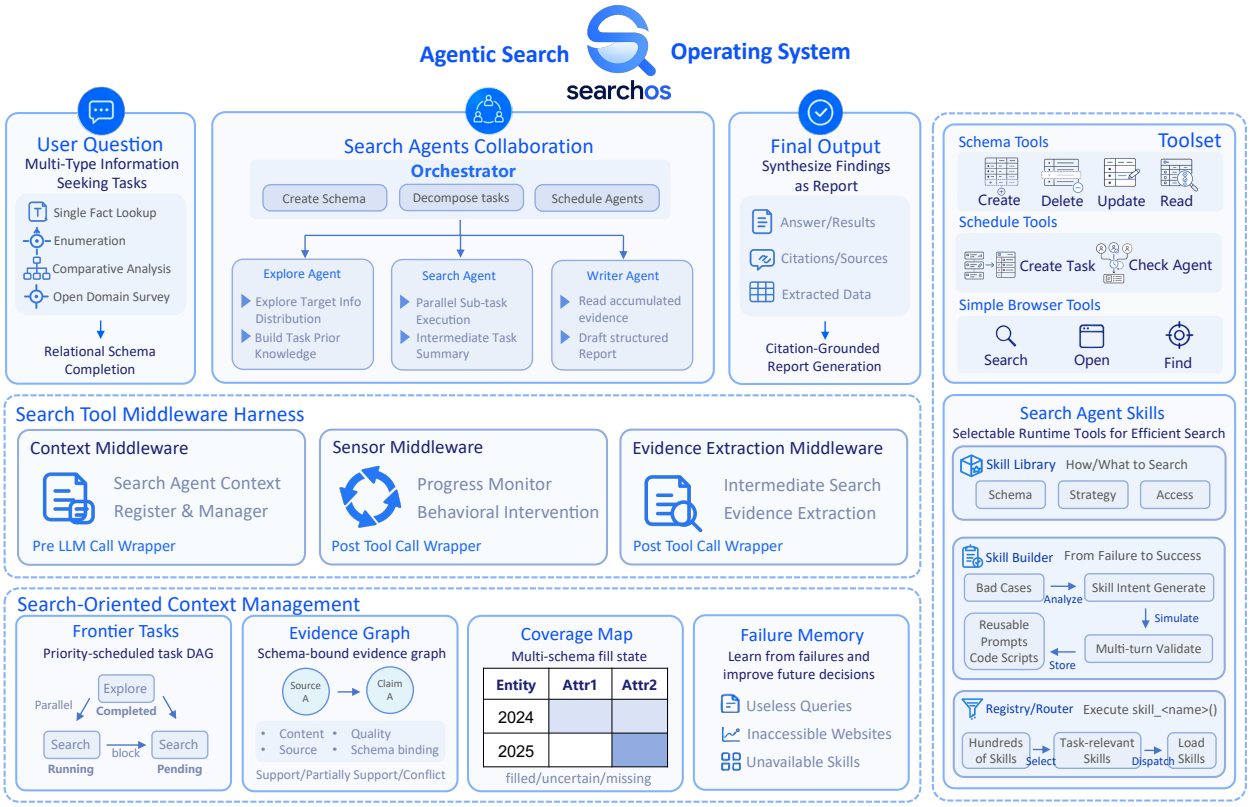


Figure 2 **SearchOS** architecture.

For each table, the system must discover a set of entities $\mathcal{E}_m = \{e_{m,i}\}_{i=1}^{N_m}$ and populate a value matrix $\mathbf{Y}_m \in \mathcal{V}^{N_m \times |\mathcal{A}_m|}$. Unlike conventional table completion, every populated value must be grounded in source evidence. We therefore maintain a citation matrix $\mathbf{C}_m$ of the same shape, where each entry links a value to a source URL and an anchored excerpt:

$$\mathcal{O} = \left\{ \left(\mathcal{E}_m, \mathbf{Y}_m, \mathbf{C}_m \right) \right\}_{m=1}^{M}. \tag{2}$$

The relational structure provides an explicit target for search, while the citation matrix makes each factual value independently verifiable. The completed schema is finally synthesized into a natural-language report with fine-grained inline citations.

3 SearchOS

To keep execution aligned with the evolving search state, we implement **SearchOS** as a stateful closed loop (Figure 2). We dispatch unresolved schema gaps through the orchestrator, commit observations through SOCM, and ground each interaction through middleware. Once complete, an optional writer produces the final report.

3.1 Search-Oriented Context Management

Because long-horizon search must track unresolved schema gaps, grounded evidence, coverage, and failed attempts, we design SOCM to externalize this information as a durable shared state

$$\mathcal{M}_t = \left(\mathcal{F}_t, \mathcal{G}_t, \mathcal{C}_t, \mathcal{W}_t \right), \tag{3}$$

where $\mathcal{F}_t$ is Frontier Task, $\mathcal{G}_t$ the Evidence Graph, $\mathcal{C}_t$ the Coverage Map, and $\mathcal{W}_t$ Failure Memory. This state resides outside agent conversations and uses locked read–modify–write updates. An agent with role r and assignment z_t receives

$$x_t^{(r)} = \phi_r(\mathcal{M}_t; z_t), \tag{4}$$

rather than the full state. We use role-specific projections to expose global gaps to the orchestrator, assigned cells to search agents, and a compact summary to the writer. Each projection is regenerated from the latest state to avoid stale model context.

3.1.1 Frontier Task

To turn schema gaps into schedulable work, we maintain Frontier Task as a dependency-aware task pool. We represent each task and the ready set as

$$f_j = (\kappa_j, s_j, p_j, B_j, Z_j, a_j, n_j), \qquad \mathcal{R}_t = \{f_j \in \mathcal{F}_t \mid s_j = \textsc{Pending} \ \wedge \ B_j \subseteq \mathcal{T}_t\}, \tag{5}$$

where κ_j, s_j, p_j, B_j, Z_j, a_j, and n_j denote type, status, priority, dependencies, target cells, assigned agent, and attempt count; $\mathcal{T}_t$ contains terminal task IDs. A task becomes pending after all dependencies terminate, then running on dispatch and completed on report collection; cycle, retry, or capacity guards may cancel it. Target-cell deduplication rejects overlapping active work, while dispatch-time revalidation closes tasks whose cells are already filled.

3.1.2 Evidence Graph

To preserve fine-grained provenance, we store atomic findings rather than page-level summaries in the Evidence Graph:

$$\begin{aligned} \mathcal{G}_t &= (\mathcal{N}_t, \mathcal{L}_t), \qquad & g_i &= (v_i, u_i, x_i, \mathbf{b}_i, \gamma_i, \tau_i, s_i), \\ \mathcal{L}_t &\subseteq \mathcal{N}_t \times \mathcal{R}_E \times \mathcal{N}_t, \qquad & \mathcal{R}_E &= \{\textsc{Support}, \textsc{Conflict}, \textsc{Refine}\}. \end{aligned} \tag{6}$$

A node $g_i \in \mathcal{N}_t$ records value v_i, source u_i, supporting span x_i, schema binding $\mathbf{b}_i = (m_i, e_i, a_i)$, confidence γ_i, provenance tier τ_i, and status s_i. Rejected or superseded nodes remain for audit but do not count toward coverage. Deduplication uses binding, normalized value, and source, preserving corroborating sources. Provenance ranks anchored spans above unanchored pages and derived summaries.

3.1.3 Coverage Map

To make progress measurable, we materialize each schema cell $c = (m, e, a)$ in the Coverage Map with status (missing, filled, uncertain, or unreachable), supporting set H_c, primary evidence, and a conflict flag. When several findings support a cell, we select

$$g_c^* = \arg\max_{g \in H_c} \big(\tau(g), \alpha(g), \gamma(g)\big), \tag{7}$$

in lexicographic order, where τ is provenance tier, α schema alignment, and γ authority-adjusted confidence. Disagreement marks a conflict rather than overwriting evidence.

Let Ω_t be the materialized cells and $\mathcal{E}_m(t)$ the discovered rows of table m. Coverage is computed as

$$\mathrm{Cov}(\mathcal{C}_t) = \frac{\sum_{c \in \Omega_t} \mathbf{1}[s(c) = \textsc{Filled}]}{|\Omega_t| + \sum_{m:\,|\mathcal{E}_m(t)|=0} |\mathcal{A}_m|}. \tag{8}$$

The denominator keeps a declared empty table unfinished. In open-set tables, entity discovery expands Ω_t, so coverage measures known rows rather than exhaustive enumeration.

3.1.4 Failure Memory

Multi-agent search requires a shared record of unsuccessful actions; otherwise, one agent may repeat a path that another has already exhausted. Failure Memory stores each record as

$$w_k = (\eta_k, \sigma_k, \chi_k, n_k, t_k), \qquad \mathcal{W}_t = \{w_k\}, \tag{9}$$

where η_k is the failure type, σ_k its task-scoped signature, χ_k the corrective guidance, n_k the recurrence count, and t_k the latest occurrence. Recorded patterns include uninformative or repeated queries, inaccessible sources, failed skills, dropped branches, and rejected claims. Matching failures increment n_k; role-specific projections expose relevant records, and recurrence strengthens "do not retry" guidance. Post-mortems from failed runs can seed memory for related tasks.

3.1.5 Atomic State Updates

SOCM updates its four memories through the same locked state interface. Evidence-producing observations jointly update the Evidence Graph and Coverage Map, while agent reports and middleware sensors update Frontier Task and Failure Memory. This coupling prevents agents from observing evidence, coverage, task status, or failure guidance from different state versions; the corresponding middleware transitions are detailed in Sections 3.3.2 and 3.3.3.

3.2 Pipeline-Parallel Multi-Agent Orchestration

Agent roles. To separate global orchestration from local search, we adopt an orchestrator–worker architecture (Wu et al., 2023). The *orchestrator* builds the schema, prioritizes gaps, and decides when to stop; *explore agents* identify candidates and sources; *search agents* collect grounded evidence for scoped gaps; and the *writer* produces a cited report. We coordinate these roles through SOCM rather than agent-to-agent conversation.

Continuous dispatch. To avoid idle time from synchronized batches, we apply the utilization benefits of pipeline parallelism (Huang et al., 2019; Narayanan et al., 2019) to role-scoped work, synchronized through SOCM. At time t, let b_t be the number of available execution slots and $\mathcal{R}_t$ the ready set defined by Frontier Task. We dispatch

$$\mathcal{D}_t = \mathrm{Top}_{\min(b_t, |\mathcal{R}_t|)}\big(\mathcal{R}_t;\, p\big), \tag{10}$$

where p is frontier priority. After each completion, we update SOCM, recompute $\mathcal{R}_t$, and refill the released slot. This event-driven policy overlaps roles when dependencies permit and avoids batch stragglers.

Toolset design. To keep each role's action space focused, we expose role-scoped toolsets (Table 1). We reserve schema and task mutation for the orchestrator, browsing for explore and search agents, skill loading for search agents, and outline and state access for the writer. Browser observations update $\mathcal{G}$ and $\mathcal{C}$ only through middleware.

Table 1 Role-scoped tools. ✓ denotes availability.

Tool Group	Orchestrator	Search	Explore	Writer
Simple Browser (search/open/find)	–	✓	✓	–
Schema & Entity CRUD	✓	–	–	–
Task Queue & Coordination	✓	–	–	–
Outline Management	–	–	–	✓
SOCM Read	–	–	–	✓
Skill Catalog (list/load)	–	✓	–	✓

Browser interface. The browser represents search as a shared navigation stack over three operations: `search` returns result pages, `open` renders and scrolls a selected source, and `find` locates keyword-matched spans. Each stack transition yields an observation; middleware determines evidence extraction and provenance. Appendix C gives the complete signatures.

3.3 Search Tool Middleware Harness

During long-horizon search, search agents face both model-level failures—such as losing task state or ignoring safeguards as context grows—and system-level disruptions, including tool errors, malformed outputs, stalled loops, and budget overruns. Because **prompt-level safeguards cannot reliably cover heterogeneous post-trained agents**, we introduce a system-level Search Tool Middleware Harness that intercepts the agent loop at model and tool boundaries. As illustrated in Figure 3, its three components form the following execution flow:

$$\widetilde{h}_t^{(r)} = \mathcal{H}_{\text{ctx}}(h_t, \mathcal{M}_t), \qquad \mathcal{M}_{t+1} = \mathcal{H}_{\text{evidence}}(o_t, \mathcal{M}_t), \qquad a_{t+1} = \mathcal{H}_{\text{sensor}}(\mathcal{M}_t, \mathcal{M}_{t+1}, \xi_t), \tag{11}$$

where $\widetilde{h}_t^{(r)}$ is the role-specific model context, o_t is a tool observation, ξ_t contains runtime counters, and a_{t+1} is a continue, correct, or stop action. Context is prepared before inference; after tool execution, evidence is grounded and committed before sensors evaluate progress and resource use.

3.3.1 Context Middleware

Before each model call, Context Middleware composes recent history h_t, shared state $\mathcal{M}_t$, and retrieved skills for role r:

$$\begin{aligned} v_t^{(r)} &= \phi_r(\mathcal{M}_t), \qquad \mathcal{K}_t = \text{TopK}\Big(\text{Retrieve}(q, \mathcal{S}, v_t^{(r)})\Big), \\ \widetilde{h}_t^{(r)} &= h_t \oplus v_t^{(r)} \oplus \psi(\mathcal{K}_t), \end{aligned} \tag{12}$$

where ϕ_r projects shared state, $\mathcal{K}_t$ is the selected skill set, ψ renders skills, and $\oplus$ composes context. The projection retains targets, grounded evidence, gaps, and failures without serializing the full history. Under budget pressure, older interactions are trimmed while state and active constraints remain.

3.3.2 Evidence Extraction Middleware

Because a visited page is not necessarily evidence, we use Evidence Extraction Middleware to extract schema-bound candidates from observation o_t:

$$\widehat{\mathcal{E}}_t = \text{Extract}(o_t, \mathcal{S}, \mathcal{C}_t), \tag{13}$$

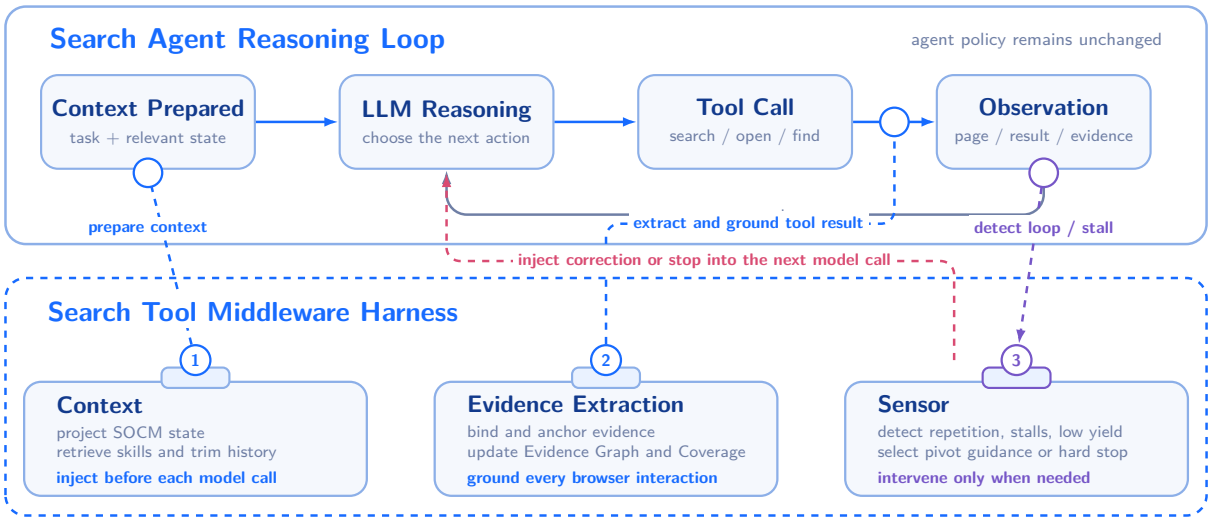


Figure 3 Illustration of middleware interventions in the Search Agent loop.

where each candidate contains an entity, attribute, value, source, and supporting span. Acceptance requires both schema binding and span anchoring:

$$\mathcal{E}_t^+ = \left\{ z \in \widehat{\mathcal{E}}_t \;\middle|\; \mathrm{Bind}(z, \mathcal{S}) = 1 \wedge \mathrm{Anchor}(z, o_t) = 1 \right\}. \tag{14}$$

We commit accepted evidence in one transition:

$$(\mathcal{G}_{t+1}, \mathcal{C}_{t+1}) = \mathcal{U}_{\text{evidence}}(\mathcal{G}_t, \mathcal{C}_t, \mathcal{E}_t^+). \tag{15}$$

This atomically records provenance and updates Coverage Map cells, making evidence capture a consequence of tool execution.

3.3.3 Sensor Middleware

To detect stalls and enforce resource limits, we use Sensor Middleware to measure coverage and evidence progress. Let $\mathrm{Cov}(\mathcal{C}_t)$ denote grounded schema coverage:

$$\Delta_t^{\text{cov}} = \mathrm{Cov}(\mathcal{C}_t) - \mathrm{Cov}(\mathcal{C}_{t-1}), \qquad \Delta_t^{\text{ev}} = |\mathcal{G}_t| - |\mathcal{G}_{t-1}|. \tag{16}$$

Over window w, the trajectory stalls when neither quantity grows:

$$s_t = \mathbb{I}\left[\sum_{j=t-w+1}^{t} \Delta_j^{\text{cov}} = 0 \;\wedge\; \sum_{j=t-w+1}^{t} \Delta_j^{\text{ev}} = 0 \right]. \tag{17}$$

Budget pressure is the maximum normalized use of iterations, searches, and time:

$$\rho_t = \max\left(\frac{n_t^{\text{iter}}}{B^{\text{iter}}}, \frac{n_t^{\text{search}}}{B^{\text{search}}}, \frac{\tau_t}{B^{\text{time}}} \right). \tag{18}$$

Sensor Middleware computes the stall state s_t and budget pressure ρ_t. The harness combines these signals with current gaps to decide whether to continue execution, inject a correction, request backfill, enter drain-only mode, or stop a branch. Because the harness enforces these transitions outside prompts, the same safeguards apply across models and roles.

(a) Frequent skill keywords.

(b) Access-skill counts and functions.

Figure 4 Pre-built skills: (a) weighted keyword frequency; (b) access-skill counts and mean functions by domain.

3.4 Hierarchical Search Skills

Because search requires reusable knowledge at different operational levels, we organize it into *orchestrator*, *strategy*, and *access* skills for global coordination, task-level methods, and source-specific retrieval. SearchOS-V1 contains 280 pre-built skills across domains (Figure 4).

Orchestrator skills. Orchestrator skills are global playbooks for task decomposition, schema and row alignment, and synthesis validation; the current skill also enforces table and row-set alignment.

Strategy skills. Strategy skills encode source-independent methods for query reformulation, entity enumeration and disambiguation, structured extraction, multi-hop and temporal reasoning, aggregation, and stall recovery.

Access skills. Access skills encode source-specific retrieval and extraction for databases, government portals, encyclopedias, company sites, and media catalogs. Some combine instructions with typed executors to avoid repeated navigation and parsing; Appendix B illustrates one for Senate.gov.

Selection and execution. At startup, we inject orchestrator skills as a shared playbook. For each task, we retrieve strategy and access skills: strategies become guidance, while executable access skills become typed tools. A query-driven router prunes the access catalog before selection.

Scope of SearchOS-V1. In SearchOS-V1, we focus on externalizing search state and providing system-level infrastructure for intermediate search artifacts. In follow-up work, we will detail large-scale search agent skill synthesis from data sources, search trajectories, and user intent. **Stay tuned!** 🥳

4 Experiments

4.1 Benchmarks

We evaluate **SearchOS** on two open-domain information-seeking benchmarks as follows:

WideSearch (Wong et al., 2025). It targets large-scale "wide" information collection: it comprises 200 manually curated questions (100 English, 100 Chinese) drawn from real user queries across more than 15 domains, each requiring an agent to gather a large set of objectively verifiable atomic facts and organize them into a complete table.

Table 2 Main results on WideSearch and GISA. The best score in each row is in **bold**, and the second-best score is underlined. Δ is the improvement of Ours over the strongest baseline.

		Single Agent		Multi-Agent System				
Benchmark	**Metric**	ReAct	Plan-and-Solve	Table-as-Search	A-MapReduce	Web2BigTable	**Ours**	Δ
WideSearch	Item · Precision	82.9	83.8	82.4	83.1	78.3	**83.9**	+0.1
	Item · Recall	70.2	72.9	73.5	74.2	73.4	**79.7**	+5.5
	Item · F1	72.9	75.2	75.4	76.0	73.8	**80.3**	+4.3
	Row · Precision	58.0	58.7	57.1	56.9	57.5	**59.0**	+0.3
	Row · Recall	48.8	50.2	51.6	49.8	54.0	**55.8**	+1.8
	Row · F1	50.9	52.2	52.7	51.4	54.5	**56.5**	+2.0
GISA	Table · Item · F1	74.8	71.2	73.4	72.5	68.1	**76.9**	+2.1
	Table · Row · F1	58.1	50.7	54.1	52.1	45.3	**59.7**	+1.6
	Set · F1	61.6	63.1	60.9	62.5	56.7	**76.5**	+13.4
	List · F1	67.1	53.8	54.2	57.4	65.5	**68.1**	+1.0
	Item · EM	0.0	16.7	16.7	33.3	**50.0**	**50.0**	0.0

GISA (Zhu et al., 2026). It evaluates general information-seeking assistants with 373 human-crafted queries reflecting authentic search scenarios; answers follow four structured formats (item, set, list, and table) that enable deterministic scoring, and the tasks couple deep multi-hop reasoning with broad cross-source aggregation.

4.2 Baselines

We compare **SearchOS** against typical baselines in two categories: *single-agent* methods, **ReAct** (Yao et al., 2022) and **Plan-and-Solve** (Wang et al., 2023), and *multi-agent* systems, **A-MapReduce** (Chen et al., 2026), **Web2BigTable** (Huang et al., 2026), and **Table-as-Search** (Lan et al., 2026).

4.3 Experimental Settings

Metrics. Following the two benchmarks, we score the assembled tables against the gold standard at two granularities. **Item**-level metrics (Precision / Recall / F1) treat each answer cell independently, while **Row**-level metrics (Precision / Recall / F1) credit a row only when *all* of its cells are correct. We additionally report **Exact Match (EM)**, the fraction of cases whose entire table exactly matches the gold answer. On GISA we break out F1 by question type (Table / Set / List / Item).

Configuration. We use `GLM-5` as the backbone for the agent roles and `Qwen3.5-35B-A3B` for Evidence Extraction. For each case we run the system three times and report the best of the three runs (Max@3), scaled by $\times 100$. Unless otherwise specified, we cap each session at 50 orchestrator iterations, 8 parallel sub-agents, 20 searches per sub-agent, and a 1800-second wall-clock budget.

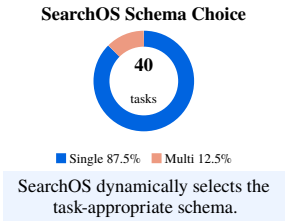
4.4 Main Results

Table 2 reports the main results. **SearchOS** achieves the best overall performance on both benchmarks, leading on all six headline F1 metrics across the two benchmarks while remaining competitive on the precision-oriented metrics.

On **WideSearch**, **SearchOS** attains the best Item-level Precision (83.9), Recall (79.7), and F1 (80.3), improving over the strongest baseline (A-MapReduce, 76.0 F1) by $+4.3$ points. The gain is concentrated in recall, consistent with the design goal of **using coverage-aware dispatch to reduce missing cells**, while precision remains the highest among all methods. On the stricter Row-level metrics,

Table 3 Relational Schema Completion analysis. Oracle means selecting the higher-Item-F1 fixed schema per case.

	Item-level			Row-level		
Setting	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
Fixed Single-Table	54.7	46.7	47.9	33.5	28.2	29.2
Fixed Multi-Table	66.9$_{(+12.2)}$	55.7$_{(+9.0)}$	58.3$_{(+10.4)}$	37.4$_{(+3.9)}$	32.6$_{(+4.4)}$	34.2$_{(+5.0)}$
Oracle Single/Multi	70.3$_{(+3.4)}$	60.7$_{(+5.0)}$	62.4$_{(+4.1)}$	44.8$_{(+7.4)}$	39.7$_{(+7.1)}$	41.2$_{(+7.0)}$
SearchOS	**76.3**$_{(+6.0)}$	**68.3**$_{(+7.6)}$	**70.6**$_{(+8.2)}$	**52.7**$_{(+7.9)}$	**47.3**$_{(+7.6)}$	**48.9**$_{(+7.7)}$



where a single wrong cell invalidates the entire row, **SearchOS** also leads in Precision (59.0), Recall (55.8), and F1 (56.5, +2.0 over the next-best Web2BigTable). Thus, the recall advantage extends to full-row consistency without sacrificing precision.

On **GISA**, **SearchOS** leads across every question type. The gains are largest on *Set* questions (76.5 vs. the best baseline's 63.1, a +13.4-point improvement), where correctly enumerating a complete answer set benefits directly from the Evidence Graph and coverage-driven completeness checks. **SearchOS** also tops *Table* F1 at both the Item (76.9) and Row (59.7) levels and *List* F1 (68.1), and matches the best Item-level EM (50.0). Overall, the results show that formulating open-domain information seeking as relational schema completion, backed by shared SOCM state, yields the most consistent gains precisely on the recall- and completeness-sensitive metrics that long-horizon search tasks demand.

5 Ablations & Analysis

In this section, we present some insightful ablation and analytical experiments to further evaluate the performance of **SearchOS**.

5.1 Fixed Schemas vs. Search-Time Schema Planning

We first test whether fixed schema structure suits all tasks. We first select 40 questions that can be decoupled into multi-table schema. For each question, we use `GPT-5.5` to construct semantically matched fixed single-table and multi-table schemas containing the same required attributes. Then we run **SearchOS** under each fixed schema and compare them with our native autonomous schema planning with explore agent.

Table 3 reports the comparison on 40 cases. Fixed multi-table is stronger on average than fixed single-table, but wins on only 21 cases, loses on 17, and ties on two by Item F1, showing that their relative advantage is task-dependent. Even an oracle that selects the better fixed schema for each case remains below **SearchOS** by 8.2 Item F1 and 7.7 Row F1 points.

Our experiments show that **different tasks favor different table structures**. Tasks with concentrated entity attributes and simple relations may be better represented by a single table, whereas those involving one-to-many relations, multiple entity types, shared attributes, or cross-entity organization are more likely to benefit from multiple tables. Across the 40 cases, **SearchOS** selected a single-table schema for 35 tasks and a multi-table schema for five. **Neither fixed structure is universally optimal.**

The consistent advantage of SearchOS shows that schema planning should remain part of the search process rather than be fixed in advance. By retaining the complete search-and-planning loop, the system can organize information according to the entity types, field distributions, repetition

patterns, and relations encountered during search, choosing a single-table, multi-table, or hybrid structure as appropriate and revising it when new evidence changes the information topology.

5.2 Pipeline-Parallel Scheduling Ablation

We next evaluate whether pipeline parallel continuous dispatch improves pipeline utilization over synchronized batches. We conduct three paired rounds with the same queries, model configuration, and concurrency limit ($K = 8$), changing only the scheduling policy on 10 WideSearch cases. Batch scheduling waits for all tasks in a group to finish, whereas continuous scheduling immediately refills each released slot.

Table 4 Round-wise scheduling results (median across 10 cases).

Round	Time Δ (↓)	Tokens Δ (↓)	Item F1 Δ (pp)
1	−32.6%	−27.7%	+2.35
2	−32.3%	−31.2%	+15.00
3	−28.6%	−10.8%	+1.85

Table 4 reports the paired results in each round. Continuous scheduling consistently reduces end-to-end time and token use while improving Item F1, showing that its advantage is stable across repeated runs.

Table 5 Pipeline scheduling ablation on WideSearch (mean across 30 trajectories per policy).

Scheduling Policy	Time (s) (↓)	Slot Utilization (↑)	Tasks/min (↑)	LLM Calls (↓)	Item F1 (↑)
Batch (Control)	629.13	34.6%	2.99	341.4	79.66
Continuous (Ours)	**476.34**	**41.7%**	**3.37**	**296.6**	**86.75**

Continuous scheduling reduces average end-to-end time by 24.3% and improves slot utilization and task throughput, while using fewer LLM calls and achieving higher Item F1. **These results show that continuous dispatch removes idle capacity at batch barriers and improves efficiency without sacrificing retrieval quality.**

5.3 Middleware-Governed Search Agent Execution Analysis

Beyond aggregate scores, we further analyze whether our Search Tool Middleware Harness governance keeps long-horizon search productive after detecting stagnation.

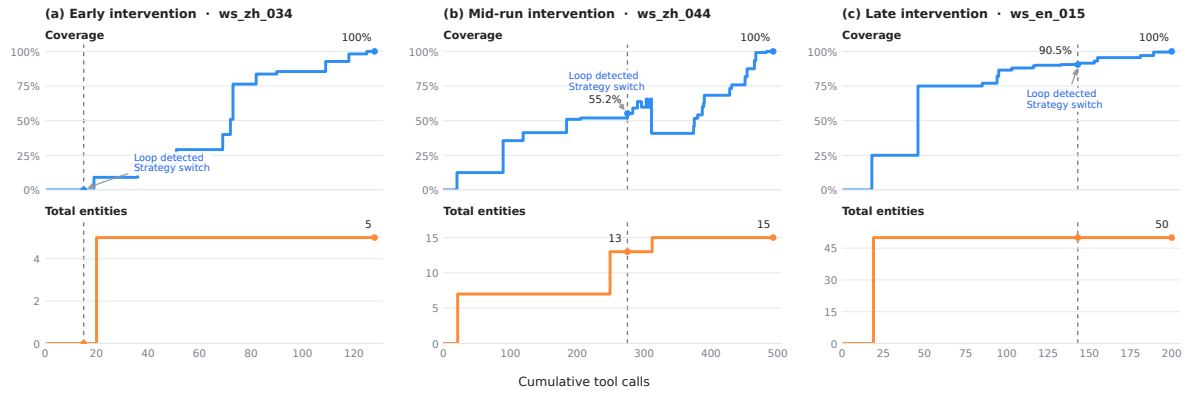


Figure 5 Middleware-governance trajectories on WideSearch.

Figure 5 presents three representative WideSearch trajectories with early, mid-run, and late interventions. In each column, the upper figure shows table coverage and the lower figure shows the cumulative number of discovered entities. The dashed line marks the point at which the Loop Sensor detects a low-progress search loop and triggers a strategy switch.

Across the different cases, interventions at early, middle, and late stages all restore search progress beyond the preceding stagnation period, although they affect coverage and entity discovery to different degrees. These trajectories provide mechanism-level evidence that the Loop Sensor can redirect stalled search toward productive work.

5.4 Skill Ablation and Efficiency

We compare **SearchOS** with and without hierarchical skills to assess their contribution to search quality and efficiency. Table 6 reports Max@3 over three runs per setting.

Table 6 Skill ablation results (Max@3).

	Item-level			Row-level		
Setting	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
Without Skills	82.2	78.5	78.3	56.0	52.4	53.1
With Skills	**83.9**	**79.7**	**80.3**	**59.0**	**55.8**	**56.5**
Δ	+1.7	+1.2	+2.0	+3.0	+3.4	+3.4

Effectiveness. Skills raise Item F1 by 2.0 points and Row F1 by 3.4 points. The larger row-level gain suggests that reusable decomposition, search, and source-access knowledge helps agents assemble coherent entity records rather than merely recover isolated cells.

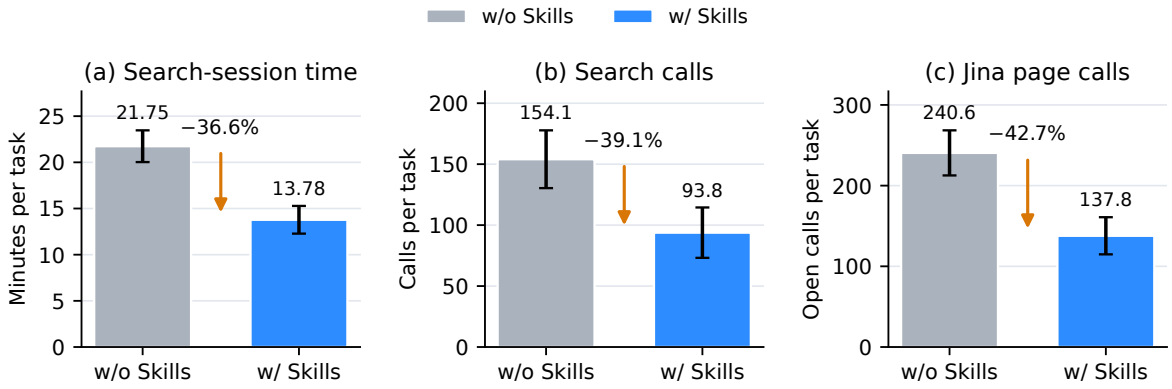


Figure 6 Efficiency with and without skills on the same 100 WideSearch questions. Error bars show 95% confidence intervals.

Efficiency. Figure 6 shows that skills reduce session time by 36.6%, search calls by 39.1%, and page calls by 42.7%. The quality gains therefore come with less exploratory trial and error rather than additional browsing. Since all skill layers are disabled together, this ablation measures their joint contribution.

6 Related Work

6.1 Agentic Information Seeking

Tool-integrated language agents interleave search, browsing, reasoning, evidence collection, and synthesis (Nakano et al., 2021; Yao et al., 2022; Trivedi et al., 2023; Qin et al., 2023; Chen et al., 2025b; Jin et al., 2025a; Li et al., 2026b; Zheng et al., 2025; Li et al., 2025). Their capabilities have advanced through search policy optimization (Qian and Liu, 2025; Luo et al., 2025), task decomposition (Chen et al., 2025b, 2026), synthetic interaction data (Tao et al., 2025; Li et al., 2025), and reinforcement-based post-training (Wang et al., 2025b; Jin et al., 2025a; Gao et al., 2025; Chen et al., 2025a; Zheng et al., 2025). Yet progress typically remains in growing interaction histories, obscuring established evidence and unresolved questions.

Broad information seeking has also been formulated as structured or table-oriented completion (Tao et al., 2025; Wong et al., 2025; Zhu et al., 2026; Lan et al., 2026; Chen et al., 2026; Huang et al., 2026). Such outputs expose missing entities and attributes and support parallel collection, but do not capture task dependencies, conflicting observations, provenance, or failed attempts. **SearchOS** instead formulates the task as *relational schema completion with grounded citations* and maintains these intermediate artifacts outside the interaction history.

6.2 Multi-Agent Orchestration Systems

Multi-agent frameworks decompose complex tasks through specialized roles, message exchange, and structured workflows (Wu et al., 2023; Hong et al., 2024; Chen et al., 2024; Qian et al., 2024; Chen et al., 2025b; Fourney et al., 2024); hierarchical approaches decouple strategic planning from specialized execution (Jin et al., 2025b), while structured execution systems distinguish dependent from parallel sub-tasks and execute independent branches concurrently (Kim et al., 2024; Zhang et al., 2025). However, coordination state usually remains in conversations, plans, or general task ledgers, making redundant collection, stale work, coverage gaps, and idle capacity difficult to manage explicitly during long-horizon search.

Recent advances in agent memory and context management address related challenges through reflection, memory stores, virtual context, compression, or trajectory consolidation (Park et al., 2023; Shinn et al., 2023; Zhong et al., 2024; Packer et al., 2023; Jiang et al., 2023; Ye et al., 2025). In contrast, Search-Oriented Context Management (SOCM) externalizes intermediate search state as a Frontier Task, Evidence Graph, Coverage Map, and Failure Memory, with role-specific projections for each agent.

6.3 Agent Harness Engineering

Recent surveys treat the agent harness as a first-class infrastructure layer governing execution, tools, context, state, lifecycle, observability, and evaluation, while engineering reports emphasize inspectable environments, feedback loops, and context curation for reliable long-horizon agents (Li et al., 2026a; Meng et al., 2026; Lopopolo, 2026; Anthropic, 2025). Prior systems instantiate parts of this layer through environment middleware, message-driven and fault-tolerant runtimes, operating-system abstractions, efficient model–tool interception, and runtime constraints (Gu et al., 2024; Gao et al., 2024; Mei et al., 2025; Abhyankar et al., 2024; Wang et al., 2025a; Ning et al., 2026; Zhang et al., 2026). The **SearchOS** Search Tool Middleware Harness applies these ideas to long-horizon information seeking by preparing role-specific context, validating and grounding evidence-producing observations, and monitoring repetition, stalls, and budget exhaustion.

7 Conclusion

In this work, we introduce **SearchOS**, a multi-agent framework that externalizes long-horizon open-domain search as system-maintained state. We formulate requests as relational schema completion with grounded citations and use Search-Oriented Context Management (SOCM) to organize coverage gaps, evidence, and intermediate search artifacts. Pipeline-parallel orchestration dispatches unresolved gaps, while our Search Tool Middleware Harness prepares context, grounds observations, and handles stalls and budget limits. A hierarchical skill library supplies reusable orchestration, search-strategy, and source-access knowledge.

Experiments on WideSearch and GISA show that **SearchOS** improves both completeness and efficiency. These results suggest that explicit search state, grounded evidence, and system-level execution control provide a practical foundation for reliable multi-agent information seeking. We will extend **SearchOS** to broader domains and multimodal settings while improving adaptation across agents, sources, and tasks.

References

R. Abhyankar, Z. He, V. Srivatsa, H. Zhang, and Y. Zhang. InferCept: Efficient intercept support for augmented large language model inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 81–95. PMLR, 2024. URL https://proceedings.mlr.press/v235/abhyankar24a.html.

Anthropic. Effective context engineering for AI agents. Anthropic Engineering, Sept. 2025. URL https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents.

P. A. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM Transactions on Database Systems (TODS)*, 1(4):277–298, 1976.

M. Chen, L. Sun, T. Li, H. Sun, Y. Zhou, C. Zhu, H. Wang, J. Z. Pan, W. Zhang, H. Chen, F. Yang, Z. Zhou, and W. Chen. Research: Learning to reason with search for llms via reinforcement learning, 2025a. URL https://arxiv.org/abs/2503.19470.

M. Chen, G. Zhang, H. Chang, Y. Guo, and S. Zhou. A-mapreduce: Executing wide search via agentic mapreduce. *arXiv preprint arXiv:2602.01331*, 2026.

W. Chen, Y. Su, J. Zuo, C. Yang, C. Yuan, C.-M. Chan, H. Yu, Y. Lu, Y.-H. Hung, C. Qian, Y. Qin, X. Cong, R. Xie, Z. Liu, M. Sun, and J. Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=EHg5GDnyq1.

Z. Chen, K. Liu, Q. Wang, J. Liu, W. Zhang, K. Chen, and F. Zhao. MindSearch: Mimicking human minds elicits deep ai searcher. In *International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=xgtXkyqw1f.

E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

A. Fourney, G. Bansal, H. Mozannar, C. Tan, E. Salinas, E. Zhu, F. Niedtner, G. Proebsting, G. Bassman, J. Gerrits, J. Alber, P. Chang, R. Loynd, R. West, V. Dibia, A. Awadallah, E. Kamar, R. Hosn, and S. Amershi. Magentic-One: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024. URL https://arxiv.org/abs/2411.04468.

D. Gao, Z. Li, X. Pan, W. Kuang, Z. Ma, B. Qian, F. Wei, W. Zhang, Y. Xie, D. Chen, L. Yao, H. Peng, Z. Zhang, L. Zhu, C. Cheng, H. Shi, Y. Li, B. Ding, and J. Zhou. AgentScope: A flexible yet robust multi-agent platform. *arXiv preprint arXiv:2402.14034*, 2024. URL https://arxiv.org/abs/2402.14034.

J. Gao, W. Fu, M. Xie, S. Xu, C. He, Z. Mei, B. Zhu, and Y. Wu. Beyond ten turns: Unlocking long-horizon agentic search with large-scale asynchronous rl, 2025. URL https://arxiv.org/abs/2508.07976.

Y. Gu, Y. Shu, H. Yu, X. Liu, Y. Dong, J. Tang, J. Srinivasa, H. Latapie, and Y. Su. Middleware for LLMs: Tools are instrumental for language agents in complex environments. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7646–7663, 2024. doi: 10.18653/v1/2024.emnlp-main.436. URL https://aclanthology.org/2024.emnlp-main.436/.

S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen. GPipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, volume 32, pages 103–112, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/093f65e080a295f8076b1c5722a46aa2-Abstract.html.

Y. Huang, Y. Chen, Z. He, Y. Chen, K. Y. Lee, H. Zhou, W. Luo, M. Fang, and J. Wang. Web2bigtable: A bi-level multi-agent llm system for internet-scale information search and extraction. *arXiv preprint arXiv:2604.27221*, 2026.

H. Jiang, Q. Wu, X. Luo, D. Li, C.-Y. Lin, Y. Yang, and L. Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*, 2023.

B. Jin, H. Zeng, Z. Yue, D. Wang, H. Zamani, and J. Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning, 2025a. URL https://arxiv.org/abs/2503.09516.

J. Jin, X. Li, G. Dong, Y. Zhang, Y. Zhu, Y. Zhao, H. Qian, and Z. Dou. Hira: A hierarchical reasoning framework for decoupled planning and execution in deep search. *arXiv preprint arXiv:2507.02652*, 2025b.

S. Kim, S. Moon, R. Tabrizi, N. Lee, M. W. Mahoney, K. Keutzer, and A. Gholami. An LLM compiler for parallel function calling. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 24370–24391. PMLR, 2024. URL https://proceedings.mlr.press/v235/kim24y.html.

T. Lan, F. Henry, B. Zhu, Q. Jia, J. Ren, Q. Pu, H. Li, L. Wang, Z. Xu, and W. Luo. Table-as-search: Formulate long-horizon agentic information seeking as table completion. *arXiv preprint arXiv:2602.06724*, 2026.

J. Li, X. Xiao, Y. Zhang, C. Liu, L. Zhao, X. Liao, Y. Ji, J. Wang, J. Gu, Y. Ge, W. Xu, X. Fang, X. Xu, T. Zhao, Y. Kim, T. Wang, J. Hamm, S. Krishnaswamy, J. Huan, and C. K. Reddy. Agent harness engineering: A survey. OpenReview preprint, 2026a. URL https://openreview.net/forum?id=eONq7FdiHa.

K. Li, Z. Zhang, H. Yin, R. Ye, Y. Zhao, L. Zhang, L. Ou, D. Zhang, X. Wu, J. Wu, X. Wang, Z. Qiao, Z. Zhang, Y. Jiang, P. Xie, F. Huang, and J. Zhou. Websailor-v2: Bridging the chasm to proprietary agents via synthetic data and scalable reinforcement learning. *CoRR*, abs/2509.13305, 2025. doi: 10.48550/ARXIV.2509.13305. URL https://doi.org/10.48550/arXiv.2509.13305.

X. Li, J. Jin, G. Dong, H. Qian, Y. Wu, J.-R. Wen, Y. Zhu, and Z. Dou. Webthinker: Empowering large reasoning models with deep research capability. *Advances in Neural Information Processing Systems*, 38:120091–120131, 2026b.

R. Lopopolo. Harness engineering: Leveraging Codex in an agent-first world. OpenAI Engineering, Feb. 2026. URL https://openai.com/index/harness-engineering/.

K. Luo, H. Qian, Z. Liu, Z. Xia, S. Xiao, S. Bao, J. Zhao, and K. Liu. Infoflow: Reinforcing search agent via reward density optimization, 2025. URL https://arxiv.org/abs/2510.26575.

K. Mei, X. Zhu, W. Xu, M. Jin, W. Hua, Z. Li, S. Xu, R. Ye, Y. Ge, and Y. Zhang. AIOS: LLM agent operating system. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=L4HHkCDz2x.

Q. Meng, Y. Wang, L. Chen, Y. Li, W. Wu, W. Jiang, Q. Wang, C. Lu, Y. Gao, Y. Wu, and Y. Hu. Agent harness for large language model agents: A survey. *Preprints*, 2026. doi: 10.20944/preprints202604.0428.v3. URL https://www.preprints.org/manuscript/202604.0428.

R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, X. Jiang, K. Cobbe, T. Eloundou, G. Krueger, K. Button, M. Knight, B. Chess, and J. Schulman. Webgpt: Browser-assisted question-answering with human feedback. *CoRR*, abs/2112.09332, 2021. URL https://arxiv.org/abs/2112.09332.

D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019. doi: 10.1145/3341301.3359646. URL https://doi.org/10.1145/3341301.3359646.

X. Ning, K. Tieu, D. Fu, T. Wei, Z. Li, Y. Bei, J. Zou, M. Ai, Z. Liu, T.-W. Li, L. Chen, Y. Zhao, K. Yang, B. Li, C. Qian, G. Li, X. Lin, Z. Zeng, R. Qiu, S. Chen, Y. Sun, X. Yang, R. Wang, R. Pan, C. Yang, D. Zhang, L. Fang, Z. Cui, Y. Cao, P. Chen, D. Sun, R. Chen, M. Srinivasan, N. Mathur, Y. Xia, H. Li, H. Yan, P. Lu, L. Zhang, T. Zhang, H. Tong, and J. He. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026. URL https://arxiv.org/abs/2605.18747.

C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, 2024. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

H. Qian and Z. Liu. Scent of knowledge: Optimizing search-enhanced reasoning with information foraging. *arXiv preprint arXiv:2505.09316*, 2025.

Y. Qin, Z. Cai, D. Jin, L. Yan, S. Liang, K. Zhu, Y. Lin, X. Han, N. Ding, H. Wang, et al. Webcpm: Interactive web search for chinese long-form question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8968–8988, 2023.

N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Z. Tao, J. Wu, W. Yin, J. Zhang, B. Li, H. Shen, K. Li, L. Zhang, X. Wang, Y. Jiang, P. Xie, F. Huang, and J. Zhou. Webshaper: Agentically data synthesizing via information-seeking formalization. *CoRR*, abs/2507.15061, 2025. doi: 10.48550/ARXIV.2507.15061. URL https://doi.org/10.48550/arXiv.2507.15061.

H. Trivedi, N. Balasubramanian, T. Khot, and A. 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 (volume 1: long papers)*, pages 10014–10037, 2023.

H. Wang, C. M. Poskitt, and J. Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. *arXiv preprint arXiv:2503.18666*, 2025a. URL https://arxiv.org/abs/2503.18666.

L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K.-W. Lee, and E.-P. 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 (ACL)*, 2023.

Z. Wang, X. Zheng, K. An, C. Ouyang, J. Cai, Y. Wang, and Y. Wu. Stepsearch: Igniting llms search ability via step-wise proximal policy optimization, 2025b. URL https://arxiv.org/abs/2505.15107.

R. Wong, J. Wang, J. Zhao, L. Chen, Y. Gao, L. Zhang, X. Zhou, Z. Wang, K. Xiang, G. Zhang, W. Huang, Y. Wang, and K. Wang. Widesearch: Benchmarking agentic broad info-seeking. *arXiv preprint arXiv:2508.07999*, 2025.

Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversation. *CoRR*, abs/2308.08155, 2023. doi: 10.48550/ARXIV.2308.08155. URL https://doi.org/10.48550/arXiv.2308.08155.

S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

R. Ye, Z. Zhang, K. Li, H. Yin, Z. Tao, Y. Zhao, L. Su, L. Zhang, Z. Qiao, X. Wang, P. Xie, F. Huang, S. Chen, J. Zhou, and Y. Jiang. Agentfold: Long-horizon web agents with proactive context management, 2025. URL https://arxiv.org/abs/2510.24699.

Y. Zhang, Z. Dou, X. Li, J. Jin, Y. Wu, Z. Li, Y. Qi, and J. Wen. Neuro-symbolic query compiler. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 12138–12155. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.findings-acl.628/.

Y. Zhang, H. Lu, J. Jin, H. Qian, S. Li, Z. Yang, Y. Zhu, J.-R. Wen, and Z. Dou. Web sitemap knowledge can enhance autonomous browsing. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 29307–29321, 2026.

Y. Zheng, D. Fu, X. Hu, X. Cai, L. Ye, P. Lu, and P. Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 414–431, 2025.

W. Zhong, L. Guo, Q. Gao, H. Ye, and Y. Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946.

Y. Zhu, X. Zhang, M. Zhang, J. Jin, L. Zhang, X. Song, K. Zhao, W. Zeng, R. Tang, H. Li, J.-R. Wen, and Z. Dou. Gisa: A benchmark for general information-seeking assistant. *arXiv preprint arXiv:2602.08543*, 2026.

A SOCM State Example

This appendix provides a concrete example of the SOCM state as injected into the orchestrator's prompt during a search session.

SOCM State Snapshot (Simplified)

== Frontier Memory ==
Active: 3/200 | Completed: 7 | Blocked: 1

ID	Question	Kind	Status
T-008	Revenue of Company X in 2024?	search	RUNNING
T-009	CEO of Company Y?	search	PENDING
T-010	Compose final report	write	BLOCKED (by T-008)

== Coverage Map ==
Table: "Fortune 500 Comparison" | Filled: 18/30 (60%)

Entity	Revenue	CEO	Founded
Company X	✓	✓	–
Company Y	✓	–	✓
Company Z	–	–	–

== Evidence Summary ==
Total nodes: 24 | Active: 21 | Conflicts: 2
Pending resolution: Company X revenue (source A: $1.2B vs source B: $1.4B)

B Executable Access Skill Case

An executable access skill is packaged as three aligned artifacts: `skill.md` provides source-specific usage guidance, `manifest.yaml` declares the typed parameter schema exposed to the agent, and `executor.py` implements dispatch, retrieval, parsing, and normalization. The following abridged case is taken from the current Senate.gov access skill.

Access Skill Case: Senate.gov (Abridged)

```
# manifest.yaml
description: Fetch current U.S. Senators by state from senate.gov
params_schema:
  function:
    type: string
    required: true
    description: one of
      [get_senators_by_state, list_all_states, get_state_history]
  state_code:
    type: string
    required: false

# typed invocation selected by the agent
{"function": "get_senators_by_state", "state_code": "OK"}

# executor.py dispatch (simplified)
async def execute(params, ctx=None):
    function = params.get("function")
    async with httpx.AsyncClient(timeout=30, follow_redirects=True) as client:
        if function == "get_senators_by_state":
            return await get_senators_by_state(client, params["state_code"])
        if function == "list_all_states":
            return await list_all_states(client)
        if function == "get_state_history":
```

```
            return await get_state_history(client, params["state_code"])

# normalized output schema
{"success": true, "state_code": "OK", "state_name": "Oklahoma",
 "senators": [{"name": ..., "party": ..., "website": ...,
               "office_address": ..., "phone": ...}],
 "senator_count": 2}
```

The manifest constrains the agent to a small operation vocabulary, while the executor hides source-specific URL construction, HTTP error handling, and BeautifulSoup parsing. Consequently, the agent receives stable structured records even though the underlying evidence is extracted from heterogeneous Senate state-profile pages.

C Implementation Reference

For reproducibility, this appendix records the tool interfaces summarized in Section 3.2: the Simple Browser operations and the orchestrator's schema, task-queue, and writer tools. These interfaces are implementation choices rather than part of the core method.

C.1 Simple Browser

Browser State Abstraction. We model the browser state as a tuple $\mathcal{B} = (\mathcal{P}, \mathcal{H}, \sigma)$, where $\mathcal{P}$ is a page cache mapping URLs to rendered page contents, $\mathcal{H}$ is a LIFO page stack recording the navigation history, and σ is a scroll cursor tracking the current viewport position within the top-of-stack page. Every browser operation that produces a new page—a search query, a page fetch, or a find operation—pushes the result onto $\mathcal{H}$, establishing a natural navigation history.

This stack-based design provides several advantages over a flat URL-based model:

- **Contextual link resolution**: When the agent issues `open(3)`, the browser walks the stack backwards to find the most recent page containing link ID 3, resolving the numeric reference unambiguously.
- **Implicit back-navigation**: The stack preserves the full navigation path, enabling further trajectory analysis.
- **Cross-page find**: `find` results reference the *source page* (the top non-find page on the stack), and subsequent `open(match_id)` calls auto-jump to the matched line, enabling efficient in-page exploration without losing context.

Browser Tools. Table 7 summarizes the three browser operations.

Table 7 Simple Browser tool specifications.

Tool	Parameters	Description
`search`	`query`: str	Web search; return URLs, titles, and snippets.
`open`	`id_or_url`: int|str; `loc`: int	Visit a page by URL or link ID, starting at a given line.
`find`	`pattern`: str|list	Grep keywords from the currently open page.

Figure 7 illustrates the rendered outputs an agent receives from `open()` and `find()` calls.

open(0)

```
[Now viewing] GPT-4 Technical Report - OpenAI
URL: https://openai.com/research/gpt-4
**viewing lines [0 - 42] of 318**

L0: # GPT-4 Technical Report
L1:
L2: [0+OpenAI+openai.com] | [1+API Docs+platform.openai.com]
L3:
L4: GPT-4 is a large multimodal model that accepts image
L5: and text inputs and produces text outputs. While less
L6: capable than humans in many real-world scenarios, it
L7: exhibits human-level performance on various professional
L8: and academic benchmarks, including passing the bar exam
L9: with a score in the top 10% of test takers
L10: [2+PDF+arxiv.org].
L11:
L12: ## Performance Highlights
L13: | Benchmark | GPT-4 Score | GPT-3.5 Score |
L14: |---------------|-------------|---------------|
L15: | Bar Exam (MBE) | 298/400 | 213/400 |
L16: | SAT Math | 700/800 | 590/800 |
L17: | GRE Quantit. | 163/170 | 147/170 |
...
```

find("bar exam")

```
[Find results] 3 matches on current page

# Match 0 -- call open(0) to jump here
  L6: capable than humans in many real-world scenarios, it
  L7: exhibits human-level performance on various professional
> L8: and academic benchmarks, including passing the bar exam
  L9: with a score in the top 10% of test takers
  L10: [2+PDF+arxiv.org].

# Match 1 -- call open(1) to jump here
  L13: | Benchmark | GPT-4 Score | GPT-3.5 Score |
  L14: |---------------|-------------|---------------|
> L15: | Bar Exam (MBE) | 298/400 | 213/400 |
  L16: | SAT Math | 700/800 | 590/800 |
  L17: | GRE Quantit. | 163/170 | 147/170 |

# Match 2 -- call open(2) to jump here
  L156: ## Appendix A: Detailed Exam Performance
  L157: On the Uniform Bar Examination, GPT-4 achieved a
> L158: scaled score of 298, placing it in the top 10% of bar
  L159: exam takers across all tested jurisdictions in 2023.
  L160: This represents a 40% improvement over GPT-3.5.
...
```

Figure 7 Browser tool outputs as seen by a search agent. **Left**: `open()` renders a page with line-numbered markdown and bracket-style link references (`[id†title†domain]`). **Right**: `find()` returns numbered matches with context; calling `open(match_id)` jumps to the source line.

C.2 Schema and Entity Management Tools

The orchestrator manages the relational search schema $\mathcal{S}$ through a suite of CRUD tools that operate on the Coverage Map (Section 3.1.3). Table 8 lists the full tool set.

Table 8 Schema and entity management tools (orchestrator only).

Tool	Key Parameters	Description
`create_schema`	`tables`, `relations`	Define schema with tables, attributes, PKs, and FKs.
`add_entities`	`entities`, `attributes`	Add rows with optional inline values (stored as evidence).
`remove_entities`	`entities`, `reason`	Soft-delete rows with mandatory justification.
`edit_entities`	`edits`	Rename PKs or edit cells. Preserves evidence refs.
`inspect_table`	`table_id`, `status`	Return row list by status (filled/partial/empty).

C.3 Task Queue and Agent Coordination Tools

The orchestrator manages the Frontier Memory (Section 3.1.1) and sub-agent lifecycle through the tools listed in Table 9.

Table 9 Task queue and agent coordination tools (orchestrator only).

Tool	Key Parameters	Description
`enqueue_tasks`	`items`	Append tasks with priority, target table, and dependencies.
`check_agents`	`timeout`: float	Block until ≥ 1 agent completes; return reports.
`stop_task`	`task_ids`, `reason`	Cancel tasks and their running sub-agents.

C.4 Writer Tools

The writer agent operates on a structured outline rather than producing free-form text, ensuring consistent organization and complete citation coverage. Table 10 lists the writer-specific tools.

Table 10 Writer agent tools for outline and section management.

Tool	Key Parameters	Description
`read_outline`	`section_id`, `toc_only`	Return TOC or full section content.
`update_outline`	`ops` (JSON array)	Batch add/remove/reorder sections.
`write_section`	`section_id`, `cited_evidence_ids`	Write section with mandatory citations.
`edit_section`	`section_id`, `old`, `new`	Exact substring replacement for surgical edits.
`annotate_section`	`section_id`, `annotation`	Attach annotation to a section.

In addition, the writer agent has read-only access to the full SOCM state and the skill catalog, enabling it to ground its writing in the latest evidence.

D Agent Trajectory Case Studies

This section presents process-level case studies from complete WideSearch trajectories. We summarize the logged decision rationale, tool action, and observable state transition at each consequential step rather than reproducing unabridged token-level reasoning. The cases were selected to expose different behaviors—parallel enrichment, scope auditing, and recovery from inaccessible sources—and are illustrative rather than aggregate evidence.

D.1 Detailed Case: Spotify 2024 Rankings

Task and outcome. Case `ws_en_022` asks for the global and U.S. top-ten songs on Spotify in 2024, enriched with artist, language, songwriter, producer, and release date. The run finishes in 362 seconds with 221 evidence nodes, 100% known-cell coverage, 97.5 Item F1, and 85.0 Row F1.

Decision trace. Table 11 reconstructs the consequential intermediate decisions from the stored trajectory. The key design choice is to establish the two authoritative ranked lists before dispatching metadata enrichment, so each sub-agent writes into a stable composite key rather than independently rediscovering row identities.

SOCM evolution. The orchestrator receives compact state snapshots while the Search Agents work. Early in the run, all 20 seeded rows have gaps. A mid-run snapshot reports only six rows with missing or uncertain columns. The final snapshot reports zero gaps, while explicitly warning that "all known rows filled" is not proof that the requested row set is exhaustive. This warning triggers the Stage-5 row-set audit rather than immediate termination.

Table 11 Condensed agent trajectory for `ws_en_022`. Rationale is summarized from logged steps; state values are taken from SOCM deltas.

Stage	Decision rationale	Action	Observable state
1. Explore	First identify an authoritative source that fixes both top-ten lists.	Dispatch one Explore Agent to inspect Spotify Wrapped and chart sources.	No schema; coverage 0%.
2. Structure	The official lists expose two row dimensions: chart category and rank.	Create one 20×8 table with primary key `[Category, Rank]`; seed both lists.	40/160 cells known; coverage 25%.
3. Parallelize	Metadata lookup is independent after row identities are fixed.	Split the 20 songs into four five-song tasks; dispatch four Search Agents with the Spotify access skill.	Four agents run concurrently.
4. Monitor	Merge completed shards without waiting for the slowest agent.	Repeatedly call `check_agents`; retain shared Coverage Map updates.	Coverage 25.0→68.8→89.4→100.0%.
5. Audit	Full known-cell coverage does not guarantee formatting consistency or scope completion.	Inspect all 20 rows; compare known row set to the two official top-ten lists.	20 expected rows present; zero missing rows.
6. Repair	Two release dates use inconsistent separators.	Apply two targeted `edit_entities` operations, then export.	20 complete rows; 221 evidence nodes.

Coverage and Governance Events (Condensed)

Moment	SOCM / middleware signal	Coverage
After schema creation	20 known rows; all 20 have missing attributes.	25.0%
First shard completes	Five rows receive title, singer, language, credits, and date evidence.	68.8%
Mid-run snapshot	20 known rows; six still have missing or uncertain attributes.	89.4%
Search stalls	Loop Sensor emits four `bare_search_without_progress` reminders across two agents.	–
Final snapshot	20 known rows; zero gaps; row-set audit still required.	100.0%

Output excerpt. The same song may occur in both charts, but the composite key preserves its category-specific rank while shared metadata is independently grounded.

Table 12 Selected final rows from `ws_en_022`.

List	Rank	Title	Singer	Release Date
Global	1	Espresso	Sabrina Carpenter	2024/04/11
Global	2	Beautiful Things	Benson Boone	2024/01/19
U.S.	1	Espresso	Sabrina Carpenter	2024/04/11
U.S.	2	Not Like Us	Kendrick Lamar	2024/05/04

D.2 Additional Process Cases

False saturation followed by scope repair (`ws_en_016`). The task enumerates Michael Phelps's individual-event medals across the Olympic Games and World Aquatics Championships. The orchestrator initially splits the work by competition family. After both agents return, known-cell coverage reaches 100%, but the table contains only 16 rows while the agent reports imply roughly 31 eligible events. Instead of treating saturation as completion, the orchestrator inspects the row set, identifies missing years and competitions, and dispatches targeted backfill tasks. The table grows to 33 and then 35 rows. A final verification pass detects a duplicate 2003 World Championships 400m individual-medley record, removes the incorrect `4:11.04` row, adds the missing 2005 100m butterfly result, and corrects medalists in several 200m freestyle rows. This trajectory illustrates why Coverage Map saturation is paired with explicit scope auditing.

Key Logged Decisions for `ws_en_016`

Observation: coverage is 100%, but only 16 rows are present; two completed agents report approximately 31 eligible medal events.
Decision: audit row scope before synthesis.
Action: inspect the table, then dispatch Olympic and World-Aquatics backfill agents.

Observation: two rows claim the 2003 400m individual medley with times `4:09.09` and `4:11.04`.
Decision: treat the duplicate identity as a verification target.
Action: verify against championship sources and remove the incorrect row.

Outcome: 35 final rows, 486 evidence nodes, 80.1 Item F1.

Blocked publisher pages and bounded recovery (`ws_en_023`). This task enumerates migration-related articles from three journals over 2020–2024. The orchestrator first corrects an ambiguous journal name, then assigns one Search Agent per journal. After the initial pass, it removes an out-of-scope 2026 article and dispatches targeted metadata tasks for five incomplete rows. Wiley and JSTOR pages repeatedly return access checks or HTTP 402 responses. The affected agents pivot to Crossref, OpenAlex, and academic-paper skills; the Loop Sensor records two explicit strategy switches for one stalled agent. When the remaining sources and per-agent budgets are exhausted, the system preserves the unresolved cells instead of fabricating metadata. Despite these access failures, the run returns 57 rows with 96.3% coverage and 93.8 Item and Row F1.

Table 13 Process summary for the three trajectory cases. Scores are scaled by $\times 100$.

Case	Primary behavior	Tasks	Evidence	Item F1	Row F1
`ws_en_022`	Parallel enrichment + final audit	5	221	97.5	85.0
`ws_en_016`	False-saturation repair + verification	8	486	80.1	28.2
`ws_en_023`	Source fallback + bounded completion	11	302	93.8	93.8

Cross-case observations. The three traces expose complementary aspects of the system that are hidden by final-answer scores:

- **Explore before committing to row identities.** In the Spotify case, an authoritative list establishes the complete key space before metadata agents are launched. This prevents independently discovered song records from drifting across the two chart categories.
- **Cell saturation and scope completion are separate conditions.** In the Phelps case, 100% coverage over 16 known rows is correctly rejected as completion because the expected event count is much larger. Coverage controls attribute filling; an explicit row-set audit controls recall.
- **Recovery remains bounded and evidence preserving.** In the journal case, sensors encourage source and tool changes after repeated failures, while resource limits eventually stop unproductive search. Unresolved cells remain visible instead of being filled from unsupported inference.

Together, these cases show the intended control loop: exploration defines a defensible search space, parallel agents accumulate evidence into shared structured state, SOCM exposes concrete residual gaps, and governance redirects or stops work according to observable progress.

E SearchOS Case Walkthroughs

The following interface snapshots are ordered by case number and then by step number. Each row contains two snapshots.

E.1 Case 1: Beijing Travel Planning

Case 1-0

Case 1-1

Case 1-2

Case 1-3

Case 1-4

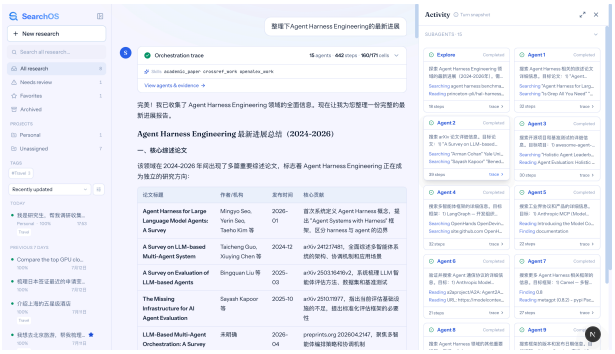
Case 1-5

Case 1-6

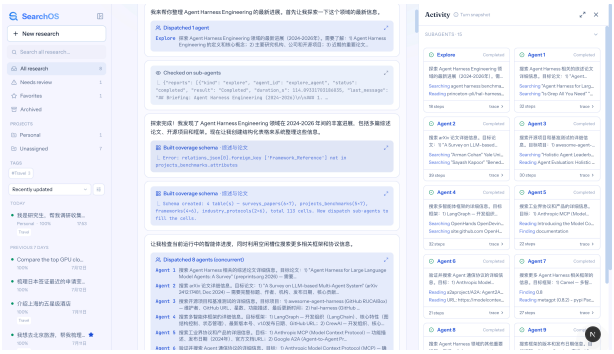
Case 1-7

Case 1-8

E.2 Case 2: Agent Harness Engineering Survey



Case 2-0

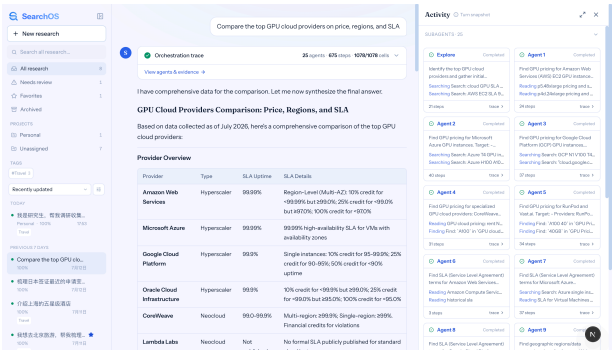


Case 2-1

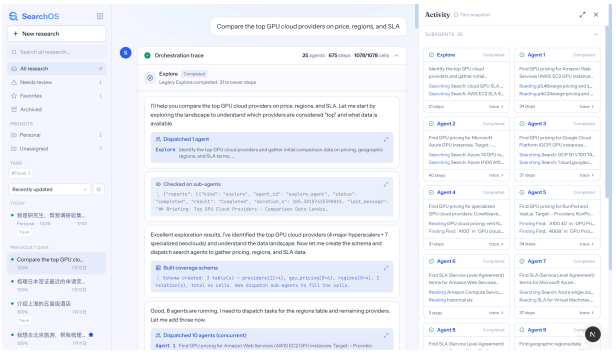
Case 2-2

Case 2-3

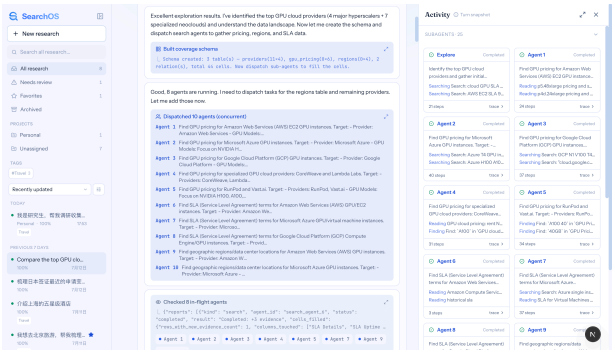
E.3 Case 3: GPU Cloud Provider Comparison



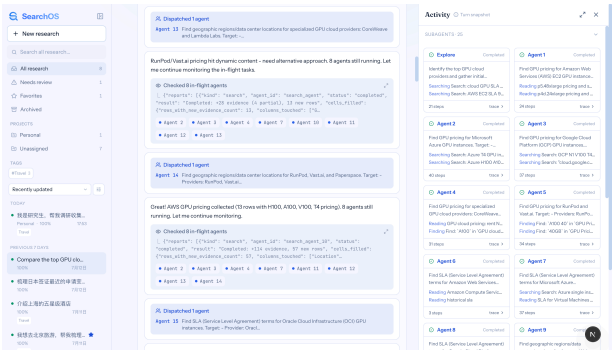
Case 3-0



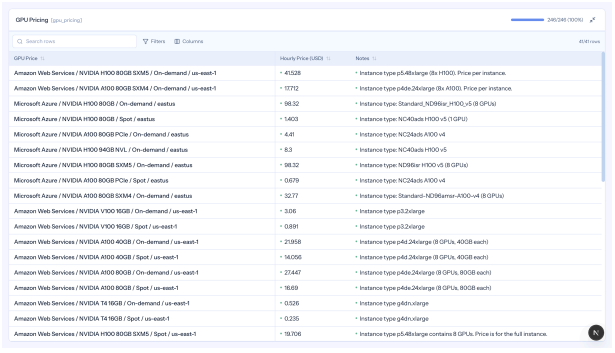
Case 3-1



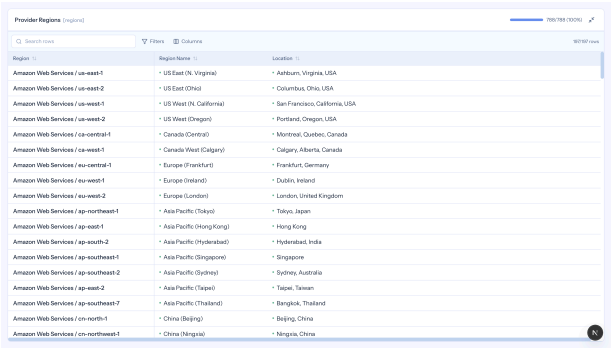
Case 3-2



Case 3-3



Case 3-4



Case 3-5