Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Web2BigTable: A Bi-Level Multi-Agent LLM System for Internet-Scale Information Search and Extraction

Meng Fang

Meng.Fang@liverpool.ac.uk

University of Liverpool

Yuxuan Huang

University of Liverpool

Yihang Chen

University College London

Zhiyuan He

Huawei Noah's Ark Lab

Yuxiang Chen

University College London

Ka Yiu Lee

Huawei Noah's Ark Lab

Huichi Zhou

University College London

Weilin Luo

Huawei Technologies

Jun Wang

University College London

Article

Keywords:

Posted Date: June 9th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9930430/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: There is **NO** Competing Interest.

Web2BigTable: A Bi-Level Multi-Agent LLM System for Internet-Scale Information Search and Extraction

Yuxuan Huang[1†], Yihang Chen[3†], Zhiyuan He[3], Yuxiang Chen[3], Ka Yiu Lee[2], Huichi Zhou[3], Weilin Luo[2], Meng Fang[1], and Jun Wang[3‡]

[1]University of Liverpool [2]Huawei Noah's Ark Lab [3]University College London

Abstract

Agentic web search increasingly faces two distinct demands: deep reasoning over a single target, and structured aggregation across many entities and heterogeneous sources. Current systems struggle on both fronts. Breadth-oriented tasks demand schema-aligned outputs with wide coverage and cross-entity consistency, while depth-oriented tasks require coherent reasoning over long, branching search trajectories. Here we introduce Web2BigTable, a multi-agent framework for web-to-table search that supports both regimes. Web2BigTable adopts a bi-level architecture in which an upper-level orchestrator decomposes the task into sub-problems and lower-level worker agents solve them in parallel. Through a closed-loop run–verify–reflect process, the framework jointly improves decomposition and execution over time via persistent, human-readable external memory, leaving the underlying language models frozen. During execution, workers coordinate through a shared workspace that makes partial findings visible, allowing them to reduce redundant exploration, reconcile conflicting evidence, and adapt to emerging coverage gaps. Web2BigTable sets a new state of the art on WideSearch, reaching an Avg@4 Success Rate of 38.50 (7.5× the second best at 5.10), Row F1 of 63.53 (+25.03 over the second best), and Item F1 of 80.12 (+14.42 over the second best). It also generalises to depth-oriented search on XBench-DeepSearch, achieving 73.0 accuracy.

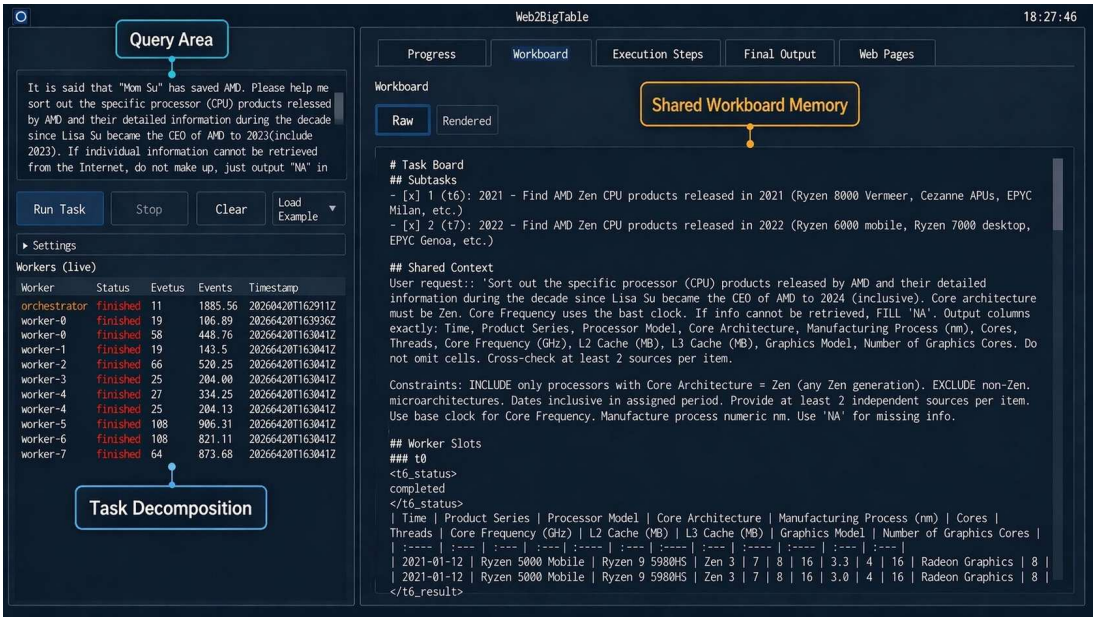


Figure 1: Web2BigTable interface during query execution. The left panel displays the user query (top left), the task decomposition status indicating the orchestrator has partitioned the query into parallel subtasks (bottom left). The right panel shows the shared workboard memory in raw Markdown view, organised into three sections: a subtask checklist tracking each worker's assignment and completion status, a shared context block containing the extraction constraints and target schema, and individual worker result slots where each worker writes its structured table output into a dedicated tag-partitioned region.

1 Introduction

Searching the open web and organising its contents into structured outputs is an important capability for large language model (LLM) agents operating over real-world information [2]. Agentic web search broadly spans two regimes. In deep search, agents iteratively retrieve, read, and reason to resolve a single complex query [12, 10]. In wide search, the goal is instead to assemble a consistent, structured view over many entities and attributes grounded in heterogeneous sources. A recent formulation of these regimes is web-to-table construction, in which an agent, given a natural-language request, must autonomously search the web and return a schema-aligned table whose rows correspond to entities and whose columns capture the requested attributes [19]. While both regimes require tool use and multi-step reasoning, wide search introduces stronger demands on coverage, decomposition, coordination, and aggregation at scale, making it a challenging and practically important setting for evaluating general web agents.

Despite recent progress, existing systems remain limited on both axes. To see the problem, compare two regimes. A query from XBench-DeepSearch [3] such as *a member of a fifth-generation Korean girl group has a brother 12 years younger; which group is it?* requires chaining indirect clues across entertainment databases, fan wikis, and biographical records to identify a single entity, which is deep search. A query from WideSearch [19] such as *list every Taylor Swift concert from 2010 to 2025 with date, city, and venue* returns hundreds of rows, each of which must be correct and mutually consistent, which is wide search. The first demands long, coherent reasoning; the second demands broad, verified coverage. A monolithic agent handles neither well at scale: the context saturates, errors compound, and the fixed plan made at the start cannot adapt to what later steps uncover. Hierarchical agent frameworks [16, 11, 31] and automatically searched workflow pipelines [29, 8] partially address this issue through task decomposition, but often rely on fixed or weakly adaptive planning strategies, with limited feedback from downstream execution to upstream decomposition [13, 27]. Meanwhile, recent self-evolving and memory-augmented agents improve execution through reusable external skills [7, 20, 17] or structured long-term memory [23, 24], yet this adaptation has largely been studied at a single level, without jointly refining how tasks are decomposed and how sub-tasks are solved. As a result, effective agentic web search requires not only decomposition, but also a mechanism to co-adapt planning and execution while enabling multiple workers to coordinate, avoid redundant exploration, reconcile conflicting evidence, and expose coverage gaps during search.

Here we introduce Web2BigTable, a multi-agent framework for web-to-table search that supports both breadth-oriented and depth-oriented instances of this task. Web2BigTable uses a bi-level architecture in which an upper-level orchestrator decomposes the task into sub-problems and lower-level worker agents solve them in parallel. The framework improves over time through a closed-loop run–verify–reflect process that jointly refines decomposition and execution, making it self-evolving through persistent, editable external memory. Concretely, the orchestrator accumulates human-readable decomposition skills that improve future task partitioning, while workers acquire reusable execution skills for retrieval, evidence verification, and intermediate synthesis. All adaptation is mediated through external memory rather than gradient updates, leaving the underlying language models frozen throughout. During execution, workers coordinate through a shared workspace that makes partial findings visible, enabling them to reduce redundant exploration, resolve conflicts, and adapt their search trajectories to the evolving global state.

We evaluate Web2BigTable on two benchmarks covering both breadth-oriented and depth-oriented web search settings. On WideSearch [19], a benchmark for broad-coverage search, Web2BigTable achieves state-of-the-art performance, reaching an Avg@4 Success Rate of 38.50 (7.5× the second best at 5.10), a Row F1 of 63.53 (+25.03 over the second best), and an Item F1 of 80.12 (+14.42 over the second best). The framework also generalises to depth-oriented search, achieving 73.0 accuracy on XBench-DeepSearch [3].

Configuration	WideSearch (Avg@4)			XBench
	SR	Row F1	Item F1	Acc.
Full system	**38.50**	**63.53**	**80.12**	**73.0**
without learned orchestrator skills	7.00	45.23	62.87	41.0
without workboard	27.50	54.81	73.45	60.0
without worker skill evolution	33.00	59.67	76.38	64.0

Table 1: **Contribution of each system component.** Removing learned orchestrator skills causes the largest drop across all metrics, confirming that bi-level strategy learning is the primary driver of performance.

2 Results

We evaluate Web2BigTable across two complementary benchmarks: WideSearch [19] for broad-coverage structured extraction, and XBench-DeepSearch [3] for deep, multi-hop reasoning. We benchmark our framework against state-of-the-art single-agent, end-to-end, and multi-agent systems, alongside comprehensive analysis to isolate the respective contributions of the learned orchestrator strategies, worker-level skill evolution, and workboard-based coordination. Implementation details of the framework, baselines, datasets, and metrics are described in Methods. Three illustrative case studies showing how learned decomposition strategies emerge from training are presented in Supplementary Notes B.1–B.3.

2.1 Performance Gain Analysis

We first conduct two sets of analysis experiments. The first isolates the contribution of each system component by disabling individual mechanisms whilst keeping all other configurations identical (Table 1). The second isolates the contribution of the framework itself from that of the underlying language models by comparing Web2BigTable against its own backbone models run as single agents (Table 2). All WideSearch results are reported as Avg@4 over four independent runs; XBench-DeepSearch results are reported as accuracy.

Learned orchestrator skills are critical. Ablating the learned decomposition skills causes a severe performance drop across both benchmarks. On WideSearch, the Success Rate falls from 38.50 to 7.00, Row F1 from 63.53 to 45.23, and Item F1 from 80.12 to 62.87; on XBench-DeepSearch, accuracy decreases from 73.0 to 41.0, a drop of 32.0 points. Without these skills, the orchestrator defaults to generic language-model-driven decomposition, which induces systematic coverage gaps in WideSearch and disjointed reasoning chains in XBench-DeepSearch. This confirms that slow-timescale orchestrator updates drive the majority of the performance advantage.

Workboard coordination enables dynamic gap recovery. Disabling the shared workboard reduces WideSearch Row F1 from 63.53 to 54.81 and XBench-DeepSearch accuracy from 73.0 to 60.0. Without shared memory, workers operate in isolation: they cannot disseminate high-quality sources, perform peer inspection, or facilitate orchestrator-triggered follow-up iterations. On WideSearch, this manifests as redundant queries and unrectified coverage gaps; on XBench-DeepSearch, intermediate findings from preliminary reasoning hops remain inaccessible to downstream agents.

Configuration	WideSearch (Avg@4)			XBench
	SR	Row F1	Item F1	Acc.
GPT-5 mini (single agent)	4.00	22.10	33.28	35.0
Gemini 3 Flash (single agent)	3.00	18.70	31.61	28.0
Web2BigTable GPT-5 mini + Gemini 3 Flash	**38.50**	**63.53**	**80.12**	**73.0**
Δ vs GPT-5 mini	+34.50	+41.43	+46.84	+38.0
Δ vs Gemini 3 Flash	+35.50	+44.83	+48.51	+45.0

Table 2: **Framework contribution vs underlying language model capability.** The same models that power Web2BigTable achieve far lower scores as single agents, confirming that the performance advantage stems from the framework design rather than model capability.

Worker skill evolution provides complementary gains. Ablating skill evolution induces a more modest decline, reducing WideSearch Row F1 to 59.67 and XBench-DeepSearch accuracy to 64.0. Constrained to their default toolset, workers lose the capacity to discover task-specific cloud skills or autonomously repair execution failures. The comparatively muted impact indicates that baseline capabilities suffice for the majority of tasks, but the persistent gap confirms that worker-level adaptation yields a consistent complementary enhancement.

Performance stems from the framework, not the underlying language models. A natural question is whether Web2BigTable's gains are attributable to the framework or simply to the choice of underlying language models. Table 2 addresses this directly. GPT-5 mini and Gemini 3 Flash, the two models powering our system, achieve only 33.28 and 31.61 Item F1 respectively as single agents. The full framework reaches 80.12 using these same models, a gain of over 46 points. Crucially, the strongest single-agent baselines in Table 3 use strictly more powerful models (Claude-4.5-Sonnet at 65.70 Item F1, GPT-5 High at 62.20) yet still fall short of Web2BigTable by at least 14 points. This confirms that the performance advantage stems from bi-level strategy learning, workboard coordination, and runtime skill evolution, not from the raw capability of the backbone language models.

2.2 Benchmark Comparison

WideSearch results. Table 3 and Figure 2 present the primary results on WideSearch. Web2BigTable achieves an Avg@4 Success Rate of 38.50 (7.5× the second best at 5.10), a Row F1 of 63.53 (+25.03 over the second best), and an Item F1 of 80.12 (+14.42 over the second best). As established in Section 2.1, these gains are attributable to the framework design rather than backbone model capability. Across all three metric categories, existing multi-agent frameworks plateau below 6 SR, 39 Row F1, and 63 Item F1 despite employing strictly stronger backbone language models (o3-high, Claude Sonnet 4). This ceiling arises because static decomposition heuristics induce systematic coverage gaps that no amount of per-worker capability can compensate for. The gains achieved by Web2BigTable are directly attributable to the learned orchestrator skills, which replace these fixed heuristics with task-adaptive decomposition strategies acquired through the run-verify-reflect pipeline.

XBench-DeepSearch results. Table 4 and Figure 3 summarise the XBench-DeepSearch evaluation. Web2BigTable achieves 73.0 accuracy, surpassing all baselines including frontier proprietary systems such as Minimax-M2 and MiroFlow (both at 72.0). Rows labelled "(OpenRouter)" correspond to our own re-evaluation of the underlying models via the OpenRouter API. XBench does not disclose its official inference configuration, including context window,

Model / System	Success Rate		Row F1		Item F1	
	Avg@4	Max@4	Avg@4	Max@4	Avg@4	Max@4
Single Agent						
Claude Sonnet 4 (Thinking)	2.30	5.00	31.70	41.90	57.90	66.70
Claude-4.5-Sonnet	–	–	–	–	65.70	–
Gemini 2.5 Pro	1.50	5.00	30.00	41.40	51.00	63.60
Gemini-3-Pro	–	–	–	–	57.00	–
OpenAI o3-high	4.50	9.00	34.00	44.10	52.60	62.30
GPT-5 High	–	–	–	–	62.20	–
GPT-5 mini	4.00	7.50	22.10	32.40	33.28	44.10
Kimi K2	1.10	3.50	29.70	41.40	54.40	65.10
Seed1.8	–	–	–	–	63.80	–
DeepSeek-R1	0.40	1.50	20.70	31.70	41.30	55.10
Doubao-1.6	2.60	5.00	30.00	44.10	48.30	63.90
Doubao-1.6-non-thinking	1.00	3.50	27.20	39.90	49.00	62.00
Gemini 3 Flash	3.00	6.00	18.70	28.50	31.61	42.30
End-to-End System						
Claude	2.50	5.00	24.10	33.50	48.40	58.50
Gemini	4.30	8.00	36.60	45.40	59.10	67.20
OpenAI o3	3.00	5.50	23.90	36.00	45.50	56.50
Multi-Agent Framework						
Claude Sonnet 4 (Thinking)	3.60	6.50	38.50	52.20	62.20	73.10
Gemini 2.5 Pro	2.00	6.50	33.50	44.60	57.40	66.30
OpenAI o3-high	5.10	9.50	37.80	50.50	57.30	68.90
Kimi K2	3.00	6.50	36.20	49.60	61.20	70.70
DeepSeek-R1	0.80	3.00	22.90	36.60	44.30	60.30
Doubao-1.6	2.50	5.50	34.00	48.90	54.60	69.70
Doubao-1.6-non-thinking	2.10	4.50	29.70	42.70	52.80	65.10
Web2BigTable (Ours) GPT-5 mini + Gemini 3 Flash	**38.50**	**40.00**	**63.53**	**65.12**	**80.12**	**82.48**

Table 3: **Performance comparison on the WideSearch benchmark.** Full results in Appendix S1.

temperature, and decoding parameters, so the 6 to 8 point gap between these rows and the officially reported scores (for example, 64.0 versus 72.0 for Minimax-M2) most likely reflects undocumented setup differences rather than capability gaps in the underlying models. We therefore treat the officially reported numbers as the primary point of comparison. The impact of the learned orchestrator skills is evident: accuracy improves from 41.0 (without learned skills) to 73.0 following strategy learning, a gain of 32.0 points from 20 synthesised training queries. This gain mirrors the pattern observed on WideSearch, where the same run-verify-reflect pipeline drives the majority of the performance advantage. The consistency of this effect across two structurally distinct benchmarks, one emphasising breadth over hundreds of entities and the other depth over multi-hop reasoning chains, validates the generalisability of the bi-level framework.

3 Discussion

We have introduced Web2BigTable, a multi-agent framework for web-to-table search that supports both broad-coverage extraction and depth-oriented multi-hop reasoning within a single architecture. The framework rests on two design decisions: a bi-level factorisation that separates query decomposition from sub-task execution, and a memory-mediated coordination protocol in which all adaptation, both within a single query and across queries, is mediated through human-readable external files rather than gradient updates. Empirically, the framework substantially outperforms single-agent, end-to-end, and multi-agent baselines on the WideSearch benchmark,

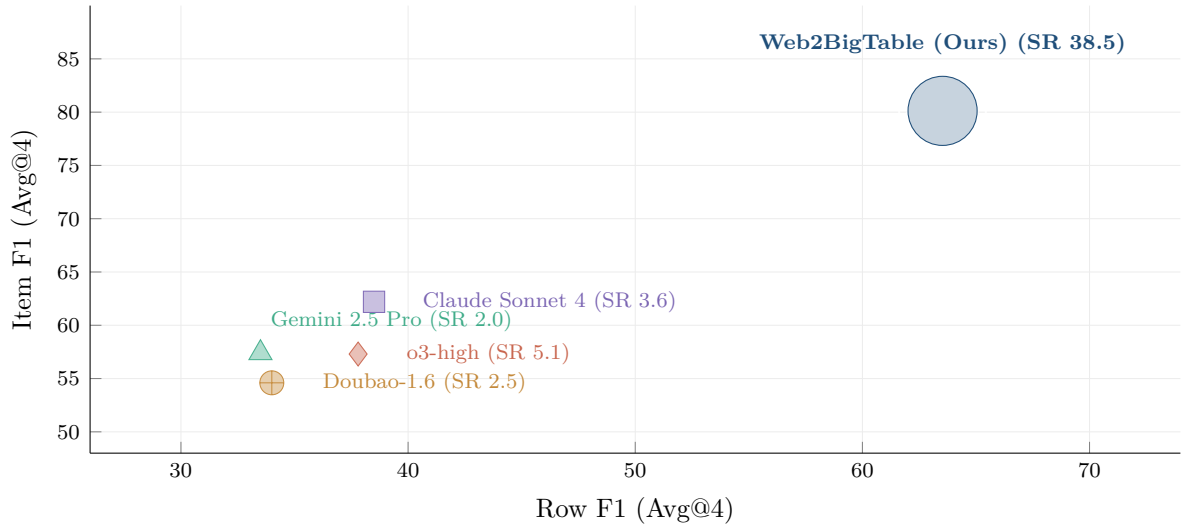


Figure 2: **Performance landscape on WideSearch (Avg@4).** Scatter points show multi-agent systems: position encodes Row F1 (x) and Item F1 (y); interior label encodes Success Rate. Web2BigTable dominates all three metrics. Full results are in Table 3.

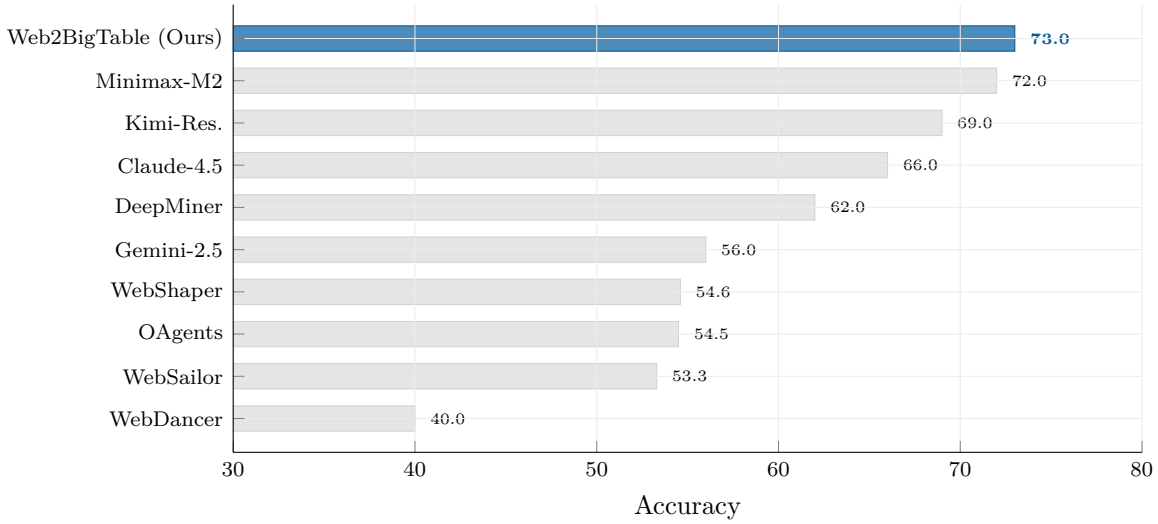


Figure 3: **Accuracy on XBench-DeepSearch.**

and reaches state-of-the-art accuracy on XBench-DeepSearch despite using lightweight backbone language models. Ablations indicate that the learned orchestrator skills account for the largest share of the gain, with workboard coordination and worker-level skill evolution providing complementary contributions.

These results bear on a tension that has shaped recent work on agentic web search. Early systems focused on single-turn retrieval to mitigate hallucinations, exemplified by WebGPT [15] and WebGLM [14], before shifting to autonomous, multi-step web navigation under benchmarks such as WebArena [33] and Mind2Web [4]. This trajectory has culminated in deep-research systems designed for exhaustive, long-horizon investigations: WebThinker [12] and Search-R1 [10] train language models to interleave reasoning and retrieval through reinforcement learning, while proprietary frameworks have demonstrated strong proficiency in multi-hop reasoning [3]. While these monolithic deep-research agents excel at vertical reasoning, they face severe limitations in large-scale structured extraction [9, 19]: when retrieving numerous entities across heterogeneous sources, single-agent architectures are bottlenecked by context saturation, error propagation,

Model / System	Accuracy
Foundation Models with Tools	
Minimax-M2	72.0
DeepSeek-V3.2	71.0
GLM-4.5	70.0
Minimax-M2 (OpenRouter)	64.0
DeepSeek-V3.2 (OpenRouter)	65.0
GLM-4.5 (OpenRouter)	64.0
Claude-4.5-Sonnet	66.0
Gemini-2.5-Pro	56.0
GPT-5 mini	35.0
Gemini 3 Flash	28.0
Deep Research System	
MiroFlow (GPT-5)	72.0
Kimi-Researcher	69.0
OAgents (Claude-3-7)	54.5
Gemini DeepResearch	50.0
OpenAI DeepResearch	26.6
Open-Source Agentic Model	
DeepMiner-32B-RL	62.0
WebShaper-32B	54.6
WebSailor-32B	53.3
AFM-32B-RL	52.0
WebDancer-QwQ	40.0
Web2BigTable (Ours) GPT-5 mini + Minimax-M2 (OpenRouter)	67.0
Web2BigTable (Ours) GPT-5 mini + DeepSeek-V3.2 (OpenRouter)	69.0
Web2BigTable (Ours) GPT-5 mini + GLM-4.5 (OpenRouter)	70.0
Web2BigTable (Ours) GPT-5 mini + Gemini 3 Flash	**73.0**

Table 4: **Performance comparison on XBench-DeepSearch.**

and rigid task decomposition. Our findings suggest that this specialisation is not architectural in nature but procedural: a bi-level architecture handles both regimes once the upper level is supplied with task-adapted decomposition strategies, discovered from training-time errors through a closed-loop verify–reflect cycle and stored as human-readable skills that can be inspected, edited, and version-controlled. The same architecture that splits a Taylor Swift discography by tour can split a multi-hop entity-resolution query by reasoning hop, provided the orchestrator has learned which structural feature to split along.

Our approach is related to but distinct from existing work on self-evolving agents, which investigates how language model agents can progressively enhance their capabilities through experiential learning without parameter updates [6]. Frameworks such as SAMULE [7] and EvolveR [20] distil transferable insights through multi-level reflection and closed-loop self-distillation, while reinforcement-learning-based approaches such as SAGE [17] and SkillRL [22] construct and exploit structured skill libraries from sequential rollouts. Other approaches facilitate continual learning via artefact-centric discovery loops [1] or by decoupling reasoning from learning using hierarchical procedural memory for compositional generalisation [5]. These systems operate at a single level: either the planner or the executor evolves, with the two layers refined in isolation.

Web2BigTable instead operates within a multi-agent paradigm in which the orchestrator evolves macro-level decomposition strategies and the parallel workers cultivate micro-level execution skills, with both processes occurring concurrently within a unified learning loop. The ablation result that orchestrator skills account for the largest share of the gain (Table 1) supports the broader claim that planning, not execution, is the dominant locus of learning in this regime, and that single-level adaptation leaves much of the available improvement on the table.

A separate consideration concerns the substrate of agent memory, which has been studied as a foundational component distinguishing autonomous agents from stateless inference [30, 21]. Recent agentic memory architectures organise memory into interconnected knowledge networks, as in A-MEM [23]; train dedicated memory managers via reinforcement learning, as in Memory-R1 [24]; or categorise experiential data through hierarchical Plan–Execute–Reflect–Memorise loops, as in MUSE [25]. Extending these concepts to multi-agent collectives, frameworks such as G-Memory [28] trace hierarchical memory across agents through structured knowledge graphs. Establishing a rigorous mathematical framework for such systems, the Memento series [18, 32] formalises memory-augmented agents through the Stateful Reflective Decision Process, providing convergence guarantees for retrieval policies operating over an evolving memory bank. Web2BigTable inherits this theoretical foundation and extends it to a hierarchical multi-agent architecture, maintaining skill memories at two distinct strata: macro-level decomposition strategies for the orchestrator, and micro-level executable skills for the workers, both refined within a unified read–write reflective learning loop. Our results provide an empirical case for this design choice: persistent, structured external memory can simultaneously support fast within-query coordination across concurrent workers and slow across-query refinement of decomposition strategies, within a single coherent system.

Several limitations qualify these results. First, the training phase requires a small set of gold-standard tables, which constrains application to domains where such supervision is available; extending the framework to settings with weaker or only partial supervision is an open question. Second, our evaluation has been confined to web search, and although the bi-level mechanism is in principle agnostic to the underlying task, its behaviour on tasks with denser inter-worker dependencies, such as collaborative writing or scientific reasoning, remains untested.

4 Methods

4.1 Problem definition

Web-to-table search task poses a qualitatively different challenge from conventional agentic web search. Whereas deep question answering converges on a single free-form response, web-to-table search demands a structured table whose rows enumerate many entities and whose cells are each anchored in live web evidence. The task is therefore inherently *wide*: a single query may span hundreds of rows, and success is governed by breadth of entity coverage rather than by depth of reasoning on any individual target.

Formally, an instance is a pair $\mathcal{T} = \langle q, \mathcal{W} \rangle$. The user query q specifies a schema of attributes to extract, for instance columns *Date, Venue, City* across hundreds of concert entries and $\mathcal{W}$ denotes the open web environment that the agent queries through retrieval and browsing tools. A valid output is a table $X \in \mathcal{X}$ whose rows enumerate the entities in q, whose columns instantiate the attributes, and whose cells each hold a value verified against web sources. Throughout this paper we adopt a scalar global utility $U(X) \in [0, 1]$ summarising overall extraction quality as the optimisation target.

Any policy π solving this task unfolds as an action–observation loop. At each step t, π samples the next action conditioned on the query q and the accumulated interaction history $h^t = (o^1, x^1, \ldots, o^t, x^t)$:

$$x^{t+1} \sim \pi(\cdot \mid q, h^t). \tag{1}$$

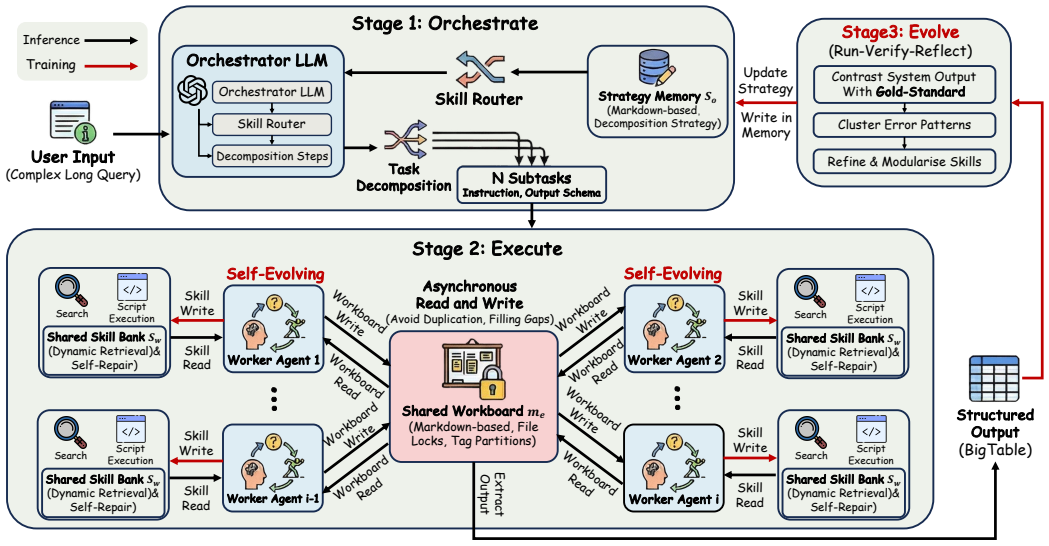


Figure 4: **Architecture of Web2BigTable, a unified training and inference framework.** The upper-level orchestrator decomposes each query into subtasks using the Orchestrator Skills $\mathcal{S}_o$, and lower-level workers execute them in parallel, drawing execution skills from the shared Worker Skills $\mathcal{S}_w$ and coordinating asynchronously through the Workboard m_e, protected by file locks and tag partitioning. Red arrows denote the additional write-back and update flows that are active only at training time, where a verify–reflect step distils each episode's trajectories into monotone updates to $\mathcal{S}_o$ and $\mathcal{S}_w$. At inference the red paths are inactive and the two skill banks are consumed read-only. Both phases operate entirely through external memory, leaving the underlying language models frozen throughout.

Each action is either a *tool-call* such as a search query or file operation that queries $\mathcal{W}$ and returns an observation o^{t+1}, the retrieved content such as a page, snippet, or tool output, appended to h^{t+1}, or a *terminal* action that emits the completed table as the policy's final response and ends the trajectory at step T. The output table is precisely $X := x^T$, the content of this terminal action.

4.2 System architecture

The architecture of Web2BigTable is shaped by two structural demands of web-to-table search. First, a single instance spans many entities under a structured schema. The single-agent policy of Equation (1) must therefore handle retrieval, state tracking, and synthesis across hundreds of rows within a bounded context window, while remaining unable to exploit the conditional independence across entities that the task naturally exhibits [12]. This motivates a factorisation of the policy into two levels: an upper-level orchestrator that decomposes the query into independent subtasks, and a lower-level pool of workers that resolves these subtasks in parallel, with the two layers strictly separated and interacting only through shared memory.

Second, the bi-level system must sustain state at two temporal scales. Concurrent workers need to exchange partial evidence within a query to avoid redundant exploration and to reconcile conflicting retrievals, and the system as a whole needs to retain the decomposition and execution skills it acquires across queries. Since the underlying language models remain frozen, such state cannot be absorbed into a single context window; it is instead externalised into two complementary memory regimes:

- **Long-term semantic memory**: a persistent store of skills that evolves only during training and is frozen at inference. It spans two levels: the upper-level orchestrator skills

$\mathcal{S}_o$, containing decomposition strategies used by the orchestrator, and the lower-level worker skills $\mathcal{S}_w$, holding execution skills shared across all workers for information seeking.

- **Short-term working memory**: a scratchpad that is transient within a single episode, realised by the workboard m_e, through which concurrent workers coordinate their ongoing trajectories.

Together, these memories replace the single-agent policy of Equation (1) with a bi-level factorisation,

$$\begin{aligned} \boldsymbol{\tau} &= (\tau_1, \ldots, \tau_N) \sim \pi_o(\cdot \mid q, \mathcal{S}_o), \\ x_i &\sim \pi_w^{(i)}(\cdot \mid \tau_i, m_e, s_i), \quad s_i \in \mathcal{S}_w, \end{aligned} \tag{2}$$

where each worker's execution skill s_i is retrieved from the shared bank $\mathcal{S}_w$ via the Memento-Skills mechanism [32]; the orchestrator selects a single decomposition skill from $\mathcal{S}_o$ through a task-router keyed on the query's structural type (Section 4.3.3). Equation (2) recasts the single-agent policy of Equation (1) through three simultaneous decompositions. First, the monolithic π splits into an upper-level orchestrator policy π_o and a pool of lower-level worker policies $\{\pi_w^{(i)}\}_{i=1}^N$: the orchestrator commits to a decomposition $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_N)$ under the orchestrator skills $\mathcal{S}_o$, and each worker i independently produces a partial output x_i conditioned on the shared workboard m_e and a retrieved skill $s_i \in \mathcal{S}_w$. Second, the global interaction history h^t that a single agent would otherwise accumulate within its context window is externalised into the shared workboard m_e and partitioned across workers as $\{h_i^t\}_i$; each worker reads only the slice of m_e relevant to its subtask, so the per-step context cost depends on subtask complexity rather than on the full table size. Third, the output X of Section 4.1 is assembled from the partial outputs as $X = (x_1, \ldots, x_N)$, and the global utility $U(X)$ becomes the optimisation target used to evolve $\mathcal{S}_o$ and $\mathcal{S}_w$ during training in Section 4.3. The two timescales are naturally separated: m_e turns over within minutes, whilst $\mathcal{S}_o$ and $\mathcal{S}_w$ accumulate experience across queries over hours or days.

The workboard m_e is not merely a message relay but a shared epistemic state. Because every worker can read the entire workboard whilst writing only to its own tagged region, the architecture makes partial findings immediately available and keeps writes non-destructive over disjoint slots, yielding a form of asynchronous consensus that scales with the worker pool whilst preserving the simplicity of a plain Markdown document. The specific adaptive behaviours this design enables are detailed in Section 4.3.4.

Memory updates. The three memory regimes evolve at different timescales through dedicated update operators. The workboard is refreshed at every inner step t as contributions from the active worker set $\mathcal{A}^t$ are merged,

$$m_e^{t+1} = \mathcal{M}_e(m_e^t, \{h_i^{t+1}\}_{i \in \mathcal{A}^t}), \tag{3}$$

with per-worker writes serialised through file locks so that no two updates collide on disjoint slots. As a short-term working memory, m_e stores the ongoing natural-language trace of a single episode, holding partial findings and intermediate observations that workers read and extend as the query unfolds, and the entire document is discarded once the episode terminates. The two skill banks evolve only across training episodes, through parallel reflect operators,

$$\mathcal{S}_o^{k+1} = \mathcal{M}_o(\mathcal{S}_o^k, r_o^{k+1}), \tag{4}$$

$$\mathcal{S}_w^{k+1} = \mathcal{M}_w(\mathcal{S}_w^k, r_o^{k+1}), \tag{5}$$

where r_o^{k+1} is the structured error report distilled from episode k's trajectories against its gold reference. Both $\mathcal{M}_o$ and $\mathcal{M}_w$ are realised as language-model-driven Run–Verify–Reflect pipelines (Section 4.3) that append new skills monotonically, without fine-tuning the underlying language

models. By contrast, $\mathcal{S}_o$ and $\mathcal{S}_w$ constitute a long-term semantic memory, stored as human-readable files that crystallise the distilled, generalised procedures extracted from completed episodes and accumulate monotonically across runs. The two memory regimes are therefore complementary rather than redundant: the workboard records what the system is currently doing and is thrown away, whereas the skill banks record what the system has learned to do and are kept.

Two-phase pipeline: training and inference. The split between long-term skill banks and short-term workboard partitions Web2BigTable's operation into two cleanly decoupled phases, separating how skills are accumulated from how they are exploited. The *training* phase (Section 4.3) runs on a small set of queries paired with gold-standard tables, distilling each episode's outcome into updates of $\mathcal{S}_o$ and $\mathcal{S}_w$ through a run–verify–reflect loop. The *inference* phase (Section 4.3.2) freezes the evolved banks $(\mathcal{S}_o^*, \mathcal{S}_w^*)$ and answers unseen user queries in a single forward pass, with no further learning. We present the two phases in their natural order: training first, to specify how the banks are acquired, followed by inference, which consumes them unchanged.

4.3 Training phase: self-evolving the skill banks

The policy in Equation (2) is only as good as the skills populating $\mathcal{S}_o$ and $\mathcal{S}_w$. Web2BigTable acquires both banks autonomously through a dedicated training phase that runs on a small split of queries paired with gold-standard tables, so that extraction quality can be evaluated objectively. At the end of training the evolved banks $(\mathcal{S}_o^*, \mathcal{S}_w^*)$ are frozen and consumed unchanged by the inference phase of Section 4.3.2.

Training objective. Training maximises the expected global utility $U(X) = \text{Item-F1}(X, X^{\text{gold}})$, a cell-level scoring function that compares the predicted table X against the gold reference X^{gold} using type-specific comparators (exact match, numeric tolerance, URL normalisation, and language-model-based semantic judgement for free text). Because subtasks are scoped to disjoint entity partitions, each worker contributes additively to U: improvements to $\mathcal{S}_o$ (yielding better decompositions) and to $\mathcal{S}_w$ (yielding better per-worker execution) both raise U without interfering, which justifies training the two skill banks jointly rather than sequentially.

Run–Verify–Reflect loop. Inside each training episode k, the system first performs one inference pass with its current skills (the inference procedure detailed in Section 4.3.2), then compares the produced table X_k against the gold reference X_k^{gold}, and finally distils the resulting error report into skill updates through two parallel reflect operators: $\mathcal{M}_o$ over orchestrator skills (Section 4.3.1) and $\mathcal{M}_w$ over worker skills (Section 4.3.2). Both updates are monotone appends of human-readable files; the underlying language models are never fine-tuned. The full procedure is summarised in Algorithm 1 and illustrated in Figure 5.

4.3.1 Orchestrator skills evolution

The reflect operator $\mathcal{M}_o$ of Equation (4) is realised as a three-stage pipeline: *Run* executes one training episode, *Verify* produces the structured error report r_o^{k+1}, and *Reflect* consumes r_o^{k+1} to synthesise new decomposition skills that append to $\mathcal{S}_o^k$. All three stages are orchestrated by language-model-driven subroutines, with no gradient update applied to the underlying models.

Run. The system processes a batch of training queries through the full pipeline, archiving structured outputs alongside per-worker JSONL trajectories that capture all tool invocations, intermediate reasoning, and temporal metadata.

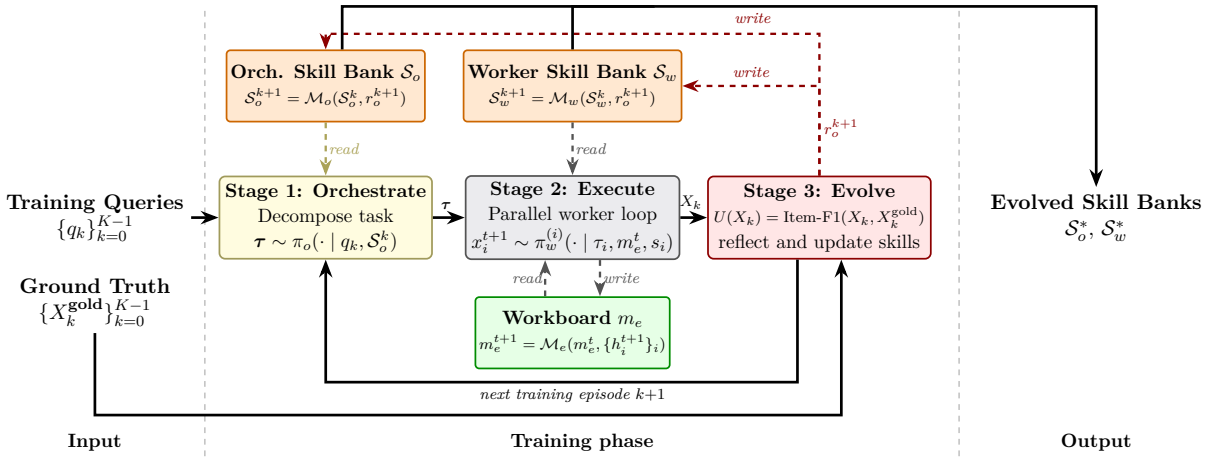


Figure 5: **Training (self-evolving) flow of Web2BigTable over one episode k.** For each training query q_k, Stage 1 reads the long-term orchestrator skills $\mathcal{S}_o$ and decomposes q_k into subtasks $\boldsymbol{\tau}$. Stage 2 dispatches these subtasks to N parallel workers, which read execution skills from $\mathcal{S}_w$ and read/write the short-term workboard m_e until convergence. Stage 3 verifies the aggregated output X_k against the gold reference, produces the structured reflection r_o^{k+1}, and consolidates it into both $\mathcal{S}_o$ (via $\mathcal{M}_o$) and $\mathcal{S}_w$ (via $\mathcal{M}_w$). Episodes are processed sequentially: the bottom black loop moves from episode k to $k{+}1$ without replanning within an episode. After K episodes, the two skill banks $(\mathcal{S}_o^*, \mathcal{S}_w^*)$ are frozen and returned as the training output, then used unchanged during inference.

Verify. Outputs are systematically evaluated against gold-standard references in three steps. First, each worker's trajectory is compressed into a structured digest by a highly efficient language model, capturing the applied decomposition strategy, search queries issued, failure points, and coverage statistics. Second, predicted outputs and gold references are parsed into row sets and compared at the cell level using type-specific scoring (exact match, numeric tolerance, URL normalisation, and language-model-based semantic judgement for free text). Third, results are aggregated into a structured error report highlighting missing row categories, low-accuracy columns, and trajectory anomalies such as context truncation or tool timeouts.

Reflect. The error report is translated into actionable skills through three sequential steps. Training queries are clustered by structural decomposition pattern rather than semantic topic, typically yielding clusters such as `split-by-entity` and `split-by-time-period`. For each cluster, a reflection language model synthesises a generalisable decomposition skill from the observed errors and successful trajectories. To rigorously prevent overfitting, these skills are constrained to use only structural placeholders. Finally, a router skill is generated that maps query characteristics to the appropriate strategy, enabling the orchestrator to process unseen queries autonomously.

All learned skills are persisted as some skill files, rendering them human-readable, editable, and version-controllable. Concretely, the reflection language model itself performs the memory write: it emits the newly synthesised decomposition and router skills directly as skill files into the skill directory backing $\mathcal{S}_o^k$, yielding $\mathcal{S}_o^{k+1}$. Prior skills are never overwritten, so updates are monotone additions. Because skills are plain text consumed via in-context learning, no parameter fine-tuning is required, and the orchestrator simply processes refined instructions during subsequent episodes, closing the slow-timescale loop.

4.3.2 Worker skills evolution

The consolidation operator $\mathcal{M}_w$ of Equation (5) is realised by a reflection language model that emits the skill changes accumulated in episode k directly into the shared bank as `SKILL.md` entries; as with $\mathcal{M}_o$, the update is a monotone append, so $\mathcal{S}_w^k \subseteq \mathcal{S}_w^{k+1}$. The runtime changes that $\mathcal{M}_w$ consolidates arise from two mechanisms operating within each episode.

Skill resolution and creation. When a specific capability is required, the `SkillResolver` executes a strictly prioritised search: (1) exact-name matching of local skills within the workspace; (2) semantic retrieval across both local and cloud-based catalogues (comprising over 8,000 pre-configured skills), employing a hybrid pipeline that integrates BM25 keyword matching with ChromaDB embedding search (BAAI/bge-m3), subsequently fused via Reciprocal Rank Fusion (RRF) and optionally refined by a cross-encoder; and (3) when no suitable match is found, the `SkillCreator` module leverages the worker's language model to synthesise a novel skill on demand, as either a function skill (an executable Python script with a command-line entry point, validated via abstract syntax tree parsing) or a knowledge skill (a structured Markdown document with YAML frontmatter). Newly synthesised skills are instantaneously indexed into the local BM25 and embedding stores.

Error-driven self-repair. Should a skill execution terminate in failure, the worker initiates an autonomous reflection loop. The error trace and the current skill code are supplied to the language model to synthesise a corrected iteration, subjected to strict abstract-syntax-tree validation before replacing the original. The same reflection loop supports extending existing skills with new capabilities via `evolve_skill`, whilst preserving backwards compatibility.

These mechanisms operate concurrently at runtime. Because the skill repository is globally shared across active workers via a singleton application context, any skill generated or amended by an individual agent is instantaneously propagated to the collective through an automated index refresh.

Algorithm 1 Web2BigTable Training (Self-Evolving)

Input: Training set $\{(q_k, X_k^{\text{gold}})\}_{k=0}^{K-1}$ of K queries paired with ground-truth tables; initial skill banks $\mathcal{S}_o^0, \mathcal{S}_w^0$; hyperparameters N (worker pool size), $T_{\max}$ (max inner steps).
Output: Evolved Skill Banks $(\mathcal{S}_o^*, \mathcal{S}_w^*)$ (consumed by Algorithm 2 at inference time).

for episode $k = 0, 1, \ldots, K-1$ **do**
  {*Stage 1 + Stage 2:* run inference with current skill banks}
  $X_k \leftarrow \textsc{Inference}(q_k; \mathcal{S}_o^k, \mathcal{S}_w^k; N, T_{\max})$ {Algorithm 2}

  {*Stage 3: Evolve*: score against gold, then consolidate into skill banks}
  Compute utility $U(X_k) \leftarrow$ Item-F1(X_k, X_k^{gold}) {cell-level training objective}
  Compress trajectories and aggregate error report $\mathcal{E}^k$
  Generate reflection $r_o^{k+1} \leftarrow \textsc{Reflect}(\mathcal{E}^k)$
  $\mathcal{S}_o^{k+1} \leftarrow \mathcal{M}_o(\mathcal{S}_o^k, r_o^{k+1})$ {Eq. (4)}
  $\mathcal{S}_w^{k+1} \leftarrow \mathcal{M}_w(\mathcal{S}_w^k, r_o^{k+1})$ {Eq. (5)}
end for
return $(\mathcal{S}_o^K, \mathcal{S}_w^K)$ {frozen skill banks $(\mathcal{S}_o^*, \mathcal{S}_w^*)$}

Inference phase: query-time execution

With the skill banks $(\mathcal{S}_o^*, \mathcal{S}_w^*)$ produced by the training phase of Section 4.3, inference runs a single forward pass over a user query q using these banks as frozen read-only inputs. No reflection, verification, or memory update is performed; the output is a structured table X. The procedure is summarised in Algorithm 2 and illustrated in Figure 6.

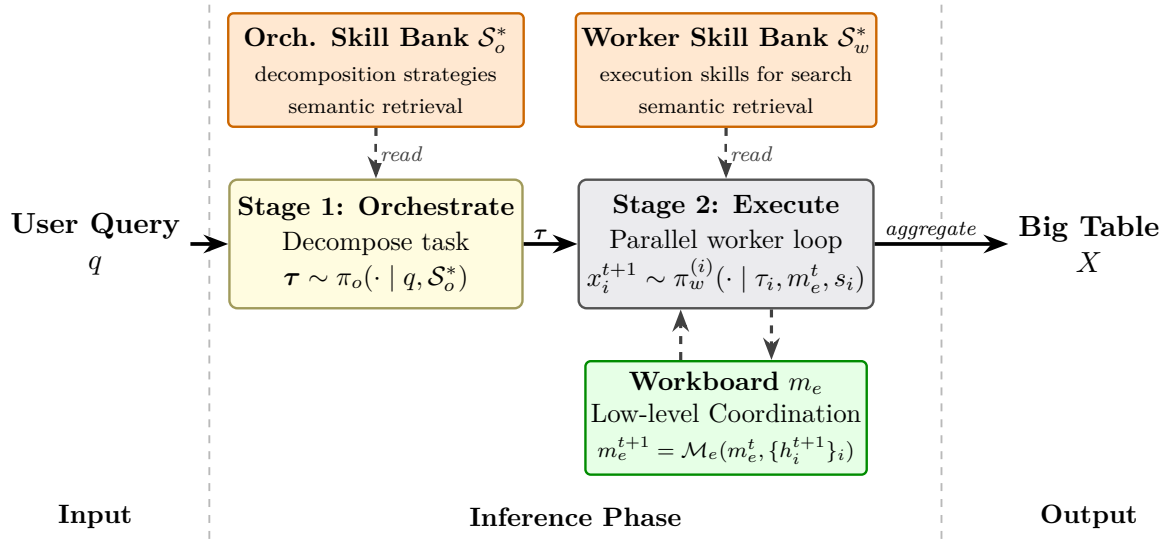


Figure 6: **Inference flow of Web2BigTable on an unseen user query q.** Using the trained skill banks $\mathcal{S}_o^*$ and $\mathcal{S}_w^*$ as frozen read-only inputs, Stage 1 decomposes q into subtasks $\boldsymbol{\tau}$. Stage 2 runs N parallel workers that resolve execution skills from $\mathcal{S}_w^*$ and coordinate through the shared workboard m_e (per-query, short-term); their partial outputs $\{x_i\}$ are aggregated into the structured big table X. No verification, reflection, or memory update is performed: the system runs a single forward pass and returns X.

4.3.3 Orchestrator

The orchestration layer governs query decomposition at runtime. Implemented as a single agent powered by a reasoning language model such as GPT-5 mini, it realises the upper level of the bi-level policy in Equation (2), partitioning the user query q into N self-contained subtasks $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_N)$ conditioned on the frozen orchestrator skills $\mathcal{S}_o^*$. It first classifies the query by invoking a `task-router` skill that evaluates structural properties (entity count, schema complexity, temporal requirements, and expected output volume), and then invokes the corresponding decomposition skill. Each resulting subtask specification includes a natural language instruction scoped to a specific data partition, the expected output schema, and a target volume of 10 to 20 items, so that the workload stays strictly within the worker's context limits.

Concurrently, the orchestrator initialises the shared workboard with a subtask checklist and tagged result slots. Once all subtasks have terminated, it performs a validation pass over row counts, column coverage, and data consistency before aggregating worker outputs into a unified table. No post-processing (summarisation, deduplication, etc.) is applied: correctness is expected to emerge directly from the decomposition design.

4.3.4 Workers

The execution layer resolves individual subtasks and coordinates concurrent workers. It realises the lower level of the bi-level policy in Equation (2) step by step: at each step t, an active worker i samples an action x_i^{t+1} conditioned on its assigned subtask τ_i, the current shared workboard m_e^t, and an execution skill s_i retrieved from the frozen worker skills $\mathcal{S}_w^*$:

$$x_i^{t+1} \sim \pi_w^{(i)}(\cdot \mid \tau_i, m_e^t, s_i). \tag{6}$$

In direct analogy with Equation (1), each x_i^{t+1} is either a tool call that queries $\mathcal{W}$ (whose observation is appended to the worker's local history h_i^t) or a response written to worker i's tagged slot on m_e. A worker terminates at the first step T at which x_i^T is a response; we set

$x_i := x_i^T$ as worker i's partial output, and the orchestrator aggregates the terminal responses into $X = (x_1, \ldots, x_N)$.

Implementation. A Model Context Protocol (MCP) server manages the worker pool, dispatching subtasks concurrently via `asyncio.gather()` with a semaphore controlling the maximum concurrency level (up to 10 workers). Each worker is an independent Memento-Skills [32] agent powered by Gemini 3 Flash, executing a ReAct loop of interleaved reasoning and tool use [26], in which the model alternates between generating a reasoning step and invoking a tool whose output is fed back as the next observation. At each step, the worker either invokes a tool or produces a final response, with generation and tool execution governed by timeouts of 120 seconds and 30 seconds, respectively.

Workers are equipped with eight built-in tools (`bash`, `str_replace`, `file_create`, `view`, `read_skill`, `route_skill`, `read_workboard`, `edit_workboard`) alongside dynamically discovered skills. Memory-intensive resources, including the BM25 index, embedding store (BAAI/bge-m3 with ChromaDB), cloud skill catalogue, and cross-encoder reranker, are shared across workers via a singleton application context to eliminate redundant loading. Each worker's complete execution trace is recorded as a JSONL trajectory, providing the foundational input for the self-evolving pipeline.

Shared workboard. The shared workboard is a structured Markdown document that instantiates the execution memory m_e. Unlike message-passing or broadcast architectures, it provides a persistent, globally visible state that all workers can read from and contribute to. It is partitioned into three sections: (i) *Task checklist*: a list of all subtasks and their completion statuses, affording every worker visibility into global progress; (ii) *Worker slots*: a tag-partitioned region where each worker writes results under a unique identifier such as `<t1_result>`, preventing write conflicts; and (iii) *Shared context*: background information, constraints, and formatting conventions pre-populated by the orchestrator.

Workers interact with the workboard through two dedicated tools: `read_workboard` for observing global progress and peer outputs, and `edit_workboard` for appending results to assigned slots. Write operations are protected by file locks and strictly confined to each worker's tagged region, whilst the entire document remains globally readable. The per-step merging of these contributions into m_e follows the update rule $\mathcal{M}_e$ specified in Equation (3).

Dynamic coordination through read-write asymmetry. This structural asymmetry is the bedrock of coordination. Because workers can freely observe peer contributions, several adaptive behaviours emerge autonomously:

- **Redundancy avoidance**: a worker observing entities already extracted by a peer skips redundant searches and refocuses on its assigned partition.
- **Coverage gap detection**: by inspecting peer outputs, an active worker can identify missing fields or inconsistencies and dynamically adjust its retrieval strategy.
- **Strategy adaptation**: workers assimilate successful patterns recorded by peers, such as highly effective source URLs, query formulations, or formatting conventions, whilst circumventing approaches that previously resulted in failures.

Workers do not merely post final answers. Their language models autonomously evaluate what intermediate information warrants sharing, such as experimental plans, partial findings, or discovered URLs, and when to write it. These continuous updates transform the workboard from a static result ledger into a live coordination medium. Because workers execute at heterogeneous speeds, this staggered dynamic creates a natural information cascade, with early-finishing workers enriching the shared context for those still running, progressively refining subsequent outputs without explicit message-passing overhead.

Algorithm 2 Web2BigTable Inference

Input: User query q; frozen skill banks $\mathcal{S}_o^*, \mathcal{S}_w^*$ from Algorithm 1; hyperparameters N (max worker pool), $T_{\max}$ (max inner steps).
Output: Big Table X.

{*Stage 1: Orchestrate*}
Classify q via `task-router` skill in $\mathcal{S}_o^*$
Sample subtasks $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_{N_q}) \sim \pi_o(\cdot \mid q, \mathcal{S}_o^*)$, with $N_q \le N$ {Eq. (2)}
Initialise workboard m_e^0 with checklist and tagged slots

{*Stage 2: Execute (asynchronous worker loop)*}
$t \leftarrow 0$
while $\neg$CONVERGED(m_e^t) **and** $t < T_{\max}$ **do**
  Identify active worker set $\mathcal{A}^t \subseteq \{1, \ldots, N_q\}$ via asynchronous dispatch
  for all worker $i \in \mathcal{A}^t$ **in parallel do**
    Read current workboard m_e^t via `read_workboard`
    Resolve skill $s_i \leftarrow$ SKILLRESOLVER$(\tau_i, \mathcal{S}_w^*)$
    Generate sub-output $x_i^{t+1} \sim \pi_w^{(i)}(\cdot \mid \tau_i, m_e^t, s_i)$ {Eq. (6)}
    Lock slot $\langle \tau_i \rangle$; write x_i^{t+1} via `edit_workboard`; **Unlock**
  end for
  $m_e^{t+1} \leftarrow \mathcal{M}_e(m_e^t, \{h_i^{t+1}\}_{i \in \mathcal{A}^t})$
  $t \leftarrow t + 1$
end while
$T \leftarrow t$

{*Aggregate and return*}
$X \leftarrow$ AGGREGATE$(\{x_i^T\}_{i=1}^{N_q})$, validated against the target table structure implicit in q
return X {big table}

4.4 Datasets

WideSearch. WideSearch [19] evaluates the reliability of language model agents in large-scale, structured information extraction from the live web. It comprises 200 manually curated tasks (100 English, 100 Chinese) spanning 15 domains. Each task requires the agent to extract multi-dimensional atomic data and organise it into a structured table. We adopt a two-phase evaluation protocol. In the *training phase*, we synthesise a set of 20 queries by perturbing task parameters (e.g., entity counts or time ranges) to avoid overlap with the original benchmark, and use these to learn the orchestrator's decomposition skills via the run–verify–reflect pipeline. In the *inference phase*, the strategy memory is frozen and the unmodified 200 tasks serve as the held-out test set.

XBench-DeepSearch. XBench-DeepSearch [3] is a professionally annotated Chinese benchmark designed to assess deep search and tool-use capabilities. Contrasting with WideSearch, which emphasises breadth, XBench-DeepSearch evaluates depth: each task necessitates multi-hop reasoning, cross-source verification, and precise answer extraction from dynamic web content. We adopt the same two-phase protocol. In the *training phase*, we generate 20 modified queries for strategy learning. In the *inference phase*, the strategy memory is frozen and the unmodified dataset serves entirely as the held-out test set.

Evaluation metrics

WideSearch. Following the WideSearch evaluation protocol, we report three complementary metrics at increasing levels of granularity. **Success Rate (SR)** is the most stringent metric, requiring a 100% match with the ground-truth table; a task is successful only if every row and cell match exactly. **Row-level F1** treats each table row as a unit, measuring the agent's ability to retrieve complete and correct records, with precision and recall computed over the matched

rows. **Item-level F1** is the most granular metric, evaluating individual cells. Cell matching employs type-specific comparators: exact match for categorical fields, numeric tolerance for quantities, URL normalisation for links, and language-model-based semantic judgement for free text. Each system is run four times independently on the test set. We report both **Avg@4** (mean across four runs) and **Max@4** (best of four runs) to capture average-case reliability and best-case capability, respectively.

XBench-DeepSearch. Following the established benchmark protocol, we report **Accuracy** evaluated via language-model-as-judge.

4.5 Model configurations and baselines

Web2BigTable configuration. Our model selection balances role-specific capability, cost, and experimental intent. The orchestrator requires planning and reasoning; the workers require fast, reliable tool use. We deliberately pair two lightweight, cost-efficient models with high rate limits to demonstrate that performance stems from framework design rather than backbone capability; stronger language models would likely yield further gains. Accordingly, the orchestrator is powered by GPT-5 mini, managing task routing, decomposition, and final synthesis. Each task is served by up to 10 parallel workers powered by Gemini 3 Flash, executing ReAct loops with access to web search, file operations, shell commands, and a shared library comprising over 8,000 cloud skills. Workers coordinate through a Markdown-based workboard protected by file locks. The orchestrator's decomposition skills are acquired through the online strategy learning pipeline (Section 4.3.1), where each completed training query triggers a three-stage cycle of verification, reflection, and skill updating. On each benchmark, skills are learned from 20 benchmark-specific training queries and frozen prior to test-time evaluation.

Baselines. On WideSearch, we compare against three categories of systems. *Single-agent baselines* include frontier language models prompted directly with the full task: Claude Sonnet 4.5, Gemini 3 Pro, GPT-5 High, and Doubao-1.6 (with and without thinking mode). *End-to-end baselines* include proprietary agentic pipelines from Claude, Gemini, and OpenAI that handle search, extraction, and synthesis as integrated products. *Multi-agent baselines* deploy the same frontier language models within a standardised orchestration framework that coordinates parallel agents via a hierarchical decomposition pipeline. On XBench-DeepSearch, baselines include foundation models with tool access (e.g., GLM-4.5, Minimax-M2), proprietary deep research systems (e.g., MiroFlow, OAgents), and open-source agentic models (e.g., DeepMiner-32B-RL, WebShaper-32B). Baseline results are sourced from published reports. For this benchmark, Web2BigTable uses 5 parallel workers rather than 10, as deep search tasks benefit more from sequential depth than parallel breadth.

References

[1] Nikolas Belle, Dakota Barnes, Alfonso Amayuelas, Ivan Bercovich, Xin Eric Wang, and William Wang. Agents of change: Self-evolving llm agents for strategic planning. *arXiv preprint arXiv:2506.04651*, 2025.

[2] William Brach, Francesco Zuppichini, Marco Vinciguerra, and Lorenzo Padoan. Scrapegraphai-100k: A large-scale dataset for llm-based web information extraction. *arXiv preprint arXiv:2602.15189*, 2026.

[3] Kaiyuan Chen, Yixin Ren, Yang Liu, et al. xbench: Tracking agents productivity scaling with profession-aligned real-world evaluations, 2025.

[4] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:64335–64366, 2023.

[5] Saman Forouzandeh, Wei Peng, Parham Moradi, Xinghuo Yu, and Mahdi Jalili. Learning hierarchical procedural memory for LLM agents through Bayesian selection and contrastive refinement. *arXiv preprint arXiv:2512.18950*, 2025. URL `https://arxiv.org/abs/2512.18950`.

[6] Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

[7] Yubin Ge, Salvatore Romeo, Jason Cai, Monica Sunkara, and Yi Zhang. SAMULE: Self-learning agents enhanced by multi-level reflection. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

[8] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL `https://arxiv.org/abs/2408.08435`.

[9] Yuxuan Huang, Yihang Chen, Haozheng Zhang, Kang Li, Huichi Zhou, Meng Fang, Linyi Yang, Xiaoguang Li, Lifeng Shang, Songcen Xu, et al. Deep research agents: A systematic examination and roadmap. *arXiv preprint arXiv:2506.18096*, 2025.

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

[11] Ka Yiu Lee, Yuxuan Huang, Zhiyuan He, Huichi Zhou, Weilin Luo, Kun Shao, Meng Fang, and Jun Wang. Infoseeker: A scalable hierarchical parallel agent framework for web information seeking. *arXiv preprint arXiv:2604.02971*, 2026.

[12] Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yutao Zhu, Yongkang Wu, Ji-Rong Wen, and Zhicheng Dou. WebThinker: Empowering large reasoning models with deep research capability. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2504.21776.

[13] Shuodi Liu, Yingzhuo Liu, Zi Wang, Yusheng Wang, Huijia Wu, Liuyu Xiang, and Zhaofeng He. Select-then-decompose: Adaptive selection strategy for task decomposition in large language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025. arXiv:2510.17922.

[14] Xiao Liu, Hanyu Lai, Hao Yu, Yifan Xu, Aohan Zuo, Yuxiao Dong, and Jie Tang. Webglm: Towards an efficient web-enhanced question answering system with human preferences. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4549–4560, 2023.

[15] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[16] Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large language model-based multi-agent collaboration. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025. arXiv:2406.07155.

[17] Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. *arXiv preprint arXiv:2512.17102*, 2025. URL `https://arxiv.org/abs/2512.17102`.

[18] Jun Wang. Memento-ii: Learning by stateful reflective memory. *arXiv preprint arXiv:2512.22716*, 2025.

[19] Ryan Wong, Jiawei Wang, Junjie Zhao, Li Chen, Yan Gao, Long Zhang, Xuan Zhou, Zuo Wang, Kai Xiang, Ge Zhang, Wenhao Huang, Yang Wang, and Ke Wang. WideSearch: Benchmarking agentic broad info-seeking. *arXiv preprint arXiv:2508.07999*, 2025.

[20] Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. EvolveR: Self-evolving LLM agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025.

[21] Shanglin Wu et al. Memory in LLM-based multi-agent systems: Mechanisms, challenges, and collective intelligence. *TechRxiv preprint*, 2025. doi: 10.36227/techrxiv.176539617.79044553. URL `https://www.techrxiv.org/doi/full/10.36227/techrxiv.176539617.79044553/v1`.

[22] Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026. URL `https://arxiv.org/abs/2602.08234`.

[23] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL `https://arxiv.org/abs/2502.12110`.

[24] Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025. URL `https://arxiv.org/abs/2508.19828`.

[25] Cheng Yang, Xuemeng Yang, Licheng Wen, Daocheng Fu, Jianbiao Mei, Rong Wu, Pinlong Cai, Yufan Shen, Nianchen Deng, Botian Shi, et al. Learning on the job: An experience-driven self-evolving agent for long-horizon tasks. *arXiv preprint arXiv:2510.08002*, 2025.

[26] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL `https://arxiv.org/abs/2210.03629`.

[27] Geunbin Yu. Adaptorch: Task-adaptive multi-agent orchestration in the era of llm performance convergence. *arXiv preprint arXiv:2602.16873*, 2026.

[28] Guibin Zhang et al. G-Memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*, 2025. URL `https://arxiv.org/abs/2506.07398`.

[29] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL `https://arxiv.org/abs/2410.10762`.

[30] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):155, 2025. doi: 10.1145/3748302. URL `https://dl.acm.org/doi/10.1145/3748302`.

[31] Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, and Jun Wang. Memento: Fine-tuning LLM agents without fine-tuning LLMs. *Preprint*, 2025.

[32] Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, et al. Memento-skills: Let agents design agents. *arXiv preprint arXiv:2603.18743*, 2026.

[33] Shuyan Zhou, Frank F Hou, Xingyao Cheng, Hongyuan Mao, Bowen Peng, Shengjia Zhong, Yihao Tai, Edward Corcodel, Rui Zhang, Uri Alon, et al. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2023.

A Detailed benchmark results

This appendix provides the per-language breakdown of WideSearch results that supports the main-text Table 3. WideSearch contains 100 English (WideSearch-en) and 100 Chinese (WideSearch-zh) tasks; results in the main text are aggregated across both subsets. Table S1 reports performance separately for each subset, demonstrating that the gains observed for Web2BigTable are consistent across languages.

B Case studies

This appendix presents three case studies illustrating how learned decomposition strategies produce qualitatively different behaviour from generic language-model-driven decomposition. Each case study contrasts the baseline (without learned orchestrator skills) against the full system (with skills learned through the run–verify–reflect pipeline).

B.1 Case A: Taylor Swift concert tours (`ws_en_006`)

This task requires compiling a comprehensive table of every concert on Taylor Swift's official tours from 1 January 2010 to 1 May 2025, with 534 ground-truth rows across five columns (Date, Concert Name, Host Country, Host City, Host Venue) spanning six tours over fifteen years.

User query. *List every concert on Taylor Swift's official tours from Jan 1, 2010 to May 1, 2025. Columns: Date, Concert Name, Host Country, Host City, Host Venue. Each show on its own row, in chronological order, no omissions.*

Baseline decomposition (without learned orchestrator skills). The orchestrator defaulted to a generic split-by-time-period strategy, dividing the 15-year span into six time windows:

- Worker 0: 2010–2011 (all tours mixed)
- Worker 1: 2012–2013
- Worker 2: 2014–2015
- Worker 3: 2016–2018
- Worker 4: 2019–2023 *(spans 5 years; the Eras Tour alone has more than 130 dates)*
- Worker 5: 2024–2025

Workers handled multiple tours mixed within each time window, and Worker 4 was overwhelmed by the five-year scope; querying "Taylor Swift concerts 2019–2023" returned noisy, unfocused results. The system retrieved 234 of 534 ground-truth rows.

Learned orchestrator skill. The task-router classified this query as `split-by-entity`. The corresponding decomposition skill (`decompose-split-by-entity/SKILL.md`), automatically synthesised from training-time error analysis, contains the following plain-Markdown rules:

```
# Split-by-entity decomposition
When the query targets a LIST OF NAMED ENTITIES (e.g., tours, brands, athletes), split
by entity name -- NOT by time period.  Each worker gets one entity as their search keyword.
Rules learned from past failures:
- If any entity has >80 expected items, split further by region.
- Always include a gap-detection worker.
- If >10% missing after gap-detection, trigger Round 2.
```

Web2BigTable decomposition (with learned skill). The system applied the learned skill, yielding eight workers split by tour name, with the Eras Tour (the largest tour) further partitioned by region:

- Worker 0: Fearless Tour (2010)
- Worker 1: Speak Now World Tour (2011–2012)
- Worker 2: The Red Tour (2013–2014)
- Worker 3: The 1989 World Tour (2015)
- Worker 4: Reputation Stadium Tour (2018)
- Worker 5: The Eras Tour: North America + early 2023
- Worker 6: The Eras Tour: International 2023–2024
- Worker 7: Gap detection and completeness check

The gap-detection worker identified incomplete Red Tour data and triggered a second round, in which two additional verification workers were dispatched. The system retrieved 653 raw rows across the six tours; after deduplication and aggregation, 556 unique rows were submitted.

Per-tour retrieval distribution (after deduplication).

Performance comparison.

B.2 Case B: AMD Zen processor catalogue (`ws_en_091`)

This task requires compiling a comprehensive table of every AMD processor with Zen-based architecture released between 2014 and 2024, with twelve columns (Time, Product Series, Processor Model, Core Architecture, Manufacturing Process, Cores, Threads, Core Frequency, L2 Cache, L3 Cache, Graphics Model, Number of Graphics Cores) and 331 ground-truth rows. The high column count makes Item F1 particularly sensitive to per-cell accuracy.

User query. *List all AMD processors with Zen architecture released from Lisa Su becoming CEO (2014) to 2024 inclusive. Columns: Time, Product Series, Processor Model, Core Architecture, Manufacturing Process (nm), Cores, Threads, Core Frequency (GHz), L2 Cache (MB), L3 Cache (MB), Graphics Model, Number of Graphics Cores. Output "NA" if information cannot be found.*

Baseline decomposition. The orchestrator divided the ten-year span into coarse time windows, mixing product lines (Ryzen Desktop, EPYC Server, Threadripper, Mobile, Embedded, PRO) within each window. Workers issued broad queries such as "AMD Zen processors 2019–2021", returning incomplete and inconsistent results across the twelve required columns. The system retrieved 137 rows; Row F1 was 18% and Item F1 was 32%. The twelve-column requirement amplified errors: even partially retrieved rows had many NA or incorrect cells across cache sizes, graphics models, and manufacturing process fields.

Learned orchestrator skill. The task-router classified this query as `split-by-category`. The decomposition skill partitions by product line rather than by year:

```
# Split-by-category decomposition
When the query covers MULTIPLE PRODUCT LINES across a long time span, split by product
category -- NOT by year.  Each worker specialises in one product line across all years.
Rules learned from past failures:
- Assign one worker per major product line (Desktop, Server, Mobile, Workstation, Embedded,
PRO).
- For product lines with >50 SKUs, split further by generation.
- Always assign a dedicated worker for cache/GPU spec lookup.
```

Web2BigTable decomposition (with learned skill). The system applied the learned skill, yielding eight workers each specialising in one product line, with a dedicated worker for cross-checking cache and GPU specifications:

- Worker 0: Ryzen Desktop (all generations)
- Worker 1: EPYC Server (Zen 1–2)
- Worker 2: EPYC Server (Zen 3–5)
- Worker 3: Ryzen Mobile (U/H/HS/HX series)
- Worker 4: Threadripper + Threadripper PRO
- Worker 5: Ryzen PRO (Desktop + Mobile)
- Worker 6: Athlon + Embedded series
- Worker 7: Spec verification (cache, GPU cores)

The system retrieved approximately 350 raw rows across eight product lines; after deduplication, 334 unique rows were submitted, achieving Row F1 of 89% and Item F1 of 96%.

Performance comparison.

B.3 Case C: ByteDance Seed and DeepSeek publications (`ws_zh_069`)

This task requires compiling research papers from two distinct source organisations (ByteDance Seed team and DeepSeek) over a 30-month window, with cross-source temporal verification against arXiv. The challenge is twofold: the two sources publish on entirely separate web platforms with different formats, and when the same paper is mirrored, the canonical date must be reconciled across publisher pages and arXiv submission records.

User query. *Compile all large-model-related papers published by the ByteDance Seed team and DeepSeek between 1 January 2023 and 30 June 2025. Search the official websites of both organisations (any paper with Seed-team participation counts). For each paper, include the publication date (yyyy-mm-dd), title, and primary authors. If two records refer to the same paper, the canonical date is the arXiv first-submit timestamp. Output a single Markdown table with columns: Organisation, Publication Date, Paper Title, Authors.*

Baseline decomposition. The orchestrator divided the 30-month window into five 6-month chunks and dispatched workers without distinguishing source organisations. Workers issued generic queries such as "ByteDance Seed papers 2024 H1" and "DeepSeek papers 2024 H2", returning fragmented results from third-party aggregators (e.g., arXiv listings) rather than the canonical source pages. The smaller-volume DeepSeek catalogue was diluted across all five workers, and arXiv-versus-publisher date conflicts were never reconciled. The system retrieved 67 rows; Row F1 was 41% and Item F1 was 58%.

The principal failure modes were threefold. First, the Seed team's roughly 120 papers were sparsely sampled because each time-window worker treated Seed as just one of many possible sources. Second, the smaller DeepSeek catalogue lost coverage entirely in early time windows where DeepSeek had no releases. Third, date inconsistencies between Seed homepage listings and arXiv records were left unresolved, contaminating the publication-date column.

Learned orchestrator skill. The task-router classified this query as `split-by-source`. The decomposition skill partitions by source organisation, with a dedicated worker for cross-source temporal verification:

```
# Split-by-source decomposition
When the query enumerates records from MULTIPLE NAMED SOURCES (e.g., specific organisations,
repositories, or publishers), assign one worker per source rather than splitting along
time or topic.

Rules learned from past failures:
- One worker per named source; never mix sources within a single worker.
- For sources with >50 records, split further by year within that source.
- If the query specifies a canonical reconciliation rule (e.g., "arXiv date takes precedence"),
assign a dedicated verification worker that cross-references the field across sources.
```

Web2BigTable decomposition (with learned skill). The system applied the learned skill, yielding seven workers split by source organisation (with the larger Seed catalogue further split by year), plus dedicated verification and normalisation workers:

- Worker 0: Seed team, 2023 (full year)
- Worker 1: Seed team, 2024 (Jan–Jun)
- Worker 2: Seed team, 2024 (Jul–Dec)
- Worker 3: Seed team, 2025 (Jan–Jun)
- Worker 4: DeepSeek, entire 30-month window
- Worker 5: arXiv cross-source date verification
- Worker 6: Author-list normalisation and duplicate detection

The verification worker flagged eight rows where the publisher-listed date diverged from the arXiv submission by more than thirty days; the orchestrator dispatched a follow-up worker to reconcile these specific entries against arXiv before final aggregation. The system retrieved 134 rows, achieving Row F1 of 91% and Item F1 of 94%.

Performance comparison.

Model / System	Success Rate		Row F1		Item F1	
	Avg@4	Max@4	Avg@4	Max@4	Avg@4	Max@4
Single Agent on WideSearch-zh						
Claude Sonnet 4 (Thinking)	0.25	1.00	30.19	39.73	53.76	63.19
Gemini 2.5 Pro	1.00	3.00	26.95	36.96	45.57	57.26
OpenAI o3-high	2.00	5.00	29.30	39.31	45.19	54.46
Kimi K2	0.25	1.00	27.79	39.03	48.81	59.64
DeepSeek-R1-0528	0.25	1.00	18.44	28.35	33.95	47.83
Doubao-1.6	1.75	4.00	29.25	42.08	43.72	58.84
Doubao-1.6-non-thinking	0.50	2.00	25.56	37.41	42.87	55.79
Single Agent on WideSearch-en						
Claude Sonnet 4 (Thinking)	4.25	9.00	33.18	44.08	62.02	70.27
Gemini 2.5 Pro	2.00	7.00	33.05	45.82	56.38	69.97
OpenAI o3-high	7.00	13.00	38.70	48.84	60.03	70.08
Kimi K2	2.00	6.00	31.54	43.68	59.91	70.52
DeepSeek-R1-0528	0.50	2.00	22.88	35.03	48.58	62.36
Doubao-1.6	3.50	6.00	30.56	46.16	52.82	68.88
Doubao-1.6-non-thinking	1.50	5.00	28.86	42.31	55.06	68.17
Multi-Agent Framework on WideSearch-zh						
Claude Sonnet 4 (Thinking)	2.75	6.00	36.85	51.46	57.13	69.53
Gemini 2.5 Pro	1.00	4.00	30.93	42.21	51.79	60.87
OpenAI o3-high	2.75	6.00	33.83	47.85	50.35	63.06
Kimi K2	1.25	3.00	34.74	48.01	56.86	66.75
DeepSeek-R1-0528	0.50	2.00	21.17	35.08	37.66	53.15
Doubao-1.6	2.25	6.00	32.83	47.49	48.79	64.43
Doubao-1.6-non-thinking	0.50	1.00	26.93	40.30	46.52	59.63
Web2BigTable (Ours)	**31.00**	**32.00**	**60.32**	**62.96**	**76.01**	**79.33**
Multi-Agent Framework on WideSearch-en						
Claude Sonnet 4 (Thinking)	4.50	7.00	40.13	52.91	67.21	76.72
Gemini 2.5 Pro	3.00	9.00	36.00	47.06	63.06	71.75
OpenAI o3-high	7.50	13.00	41.78	53.20	64.27	74.80
Kimi K2	4.75	10.00	37.71	51.20	65.44	74.68
DeepSeek-R1-0528	1.00	4.00	24.57	38.10	50.91	67.54
Doubao-1.6	2.75	5.00	35.14	50.38	60.39	74.87
Doubao-1.6-non-thinking	3.75	8.00	32.38	44.99	58.97	70.62
Web2BigTable (Ours)	**46.00**	**48.00**	**66.74**	**67.28**	**84.23**	**85.63**
End-to-End Systems on WideSearch-zh						
Claude	0.00	0.00	20.84	28.92	43.51	52.14
Gemini	1.50	4.00	32.32	40.52	52.92	60.44
OpenAI o3	3.00	5.00	27.40	38.34	46.03	56.51
End-to-End Systems on WideSearch-en						
Claude	5.00	10.00	27.39	38.07	53.29	64.81
Gemini	7.00	12.00	40.95	50.29	65.18	73.90
OpenAI o3	3.00	6.00	20.42	33.72	45.02	56.47

Table S1: **Detailed per-language results on the WideSearch benchmark.** Each system is run four times independently on the 100-task English subset and the 100-task Chinese subset. Avg@4 and Max@4 denote the mean and maximum over the four runs. Web2BigTable outperforms all baselines on both language subsets across all three metric categories.

Tour	Rows	Tour	Rows
The Red Tour (2013–2014)	142	Fearless Tour (2010)	85
The Eras Tour (2023–2024)	118	The 1989 World Tour (2015)	73
Speak Now World Tour (2011–2012)	93	Reputation Stadium Tour (2018)	45
Total (deduplicated)			**556**

Table S2: Per-tour retrieval distribution for Case A after deduplication.

	Row F1	Item F1	Rows
GPT-5 mini (single agent)	15.2%	71.3%	81
Gemini 3 Flash (single agent)	12.8%	68.5%	64
Web2BigTable (without learned skills)	26.8%	79.7%	234
Web2BigTable (full)	**93.8%**	**96.2%**	**556**
Δ (full vs best single agent)	+78.6	+24.9	+475
Δ (full vs without skills)	+67.0	+16.5	+322

Table S3: Per-task performance for Case A. GPT-5 mini and Gemini 3 Flash retrieve fewer than 100 rows as single agents; the same two models inside Web2BigTable retrieve 556 deduplicated rows at 93.8% Row F1, indicating that the gain stems from the framework rather than the underlying language models.

	Row F1	Item F1	Rows
GPT-5 mini (single agent)	10.5%	25.3%	48
Gemini 3 Flash (single agent)	8.2%	21.7%	35
Web2BigTable (without learned skills)	18%	32%	137
Web2BigTable (full)	**89%**	**96%**	**334**
Δ (full vs best single agent)	+78.5	+70.7	+286
Δ (full vs without skills)	+71.0	+64.0	+197

Table S4: Per-task performance for Case B. With twelve columns per row, single agents retrieve fewer than 50 rows and fill barely 20% of cells correctly. The learned product-line decomposition enables each worker to query authoritative sources (e.g., AMD product pages, WikiChip) for its specific category, while a dedicated spec-verification worker cross-checks cache and GPU fields.

	Row F1	Item F1	Rows
GPT-5 mini (single agent)	22.4%	39.6%	29
Gemini 3 Flash (single agent)	18.7%	34.2%	23
Web2BigTable (without learned skills)	41%	58%	67
Web2BigTable (full)	**91%**	**94%**	**134**
Δ (full vs best single agent)	+68.6	+54.4	+105
Δ (full vs without skills)	+50.0	+36.0	+67

Table S5: Per-task performance for Case C. The two sources have asymmetric catalogue sizes (approximately 120 versus 10 papers). Single agents retrieve fewer than 30 rows, missing the majority of Seed papers and most DeepSeek entries entirely. Source-based decomposition assigns dedicated workers per organisation, while a verification worker resolves arXiv-versus-publisher date conflicts that all other configurations silently ignore.