MULTI-AGENT WORLD: SCALING MULTI-AGENT ORCHESTRATION TRAINING VIA RECURSIVE ENVIRONMENT IMPROVEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems assign complementary subtasks to different agents, and their effectiveness depends on an orchestrator that partitions a request into subtasks, schedules them against their dependencies, and integrates the evidence that workers return. Training such an orchestrator requires multi-agent environments, in which a task provides subtasks that can be distributed across agents and a verifier that reports how they were distributed. Existing agentic environments are synthesized for a single agent, and they yield mostly linear call chains under outcome-only verification. An orchestrator trained inside them therefore has no scheduling decision to learn and no feedback on the decomposition it chose. We present *Multi-Agent World* (MAW), an end-to-end framework that develops multi-agent orchestration by coupling multi-agent environment synthesis with multi-agent reinforcement learning. MAW introduces *recursive environment improvement*, which recursively composes reference task graphs through serial and parallel operators inside single-agent environments, and turns each composed graph into one coherent request through semantic reconstruction and execution-grounded admission. The same composition record identifies which subtasks are verified as independent, which is what a *schedule-aware process reward* needs in order to score the trajectory of the whole team. Trained under these process rewards, the orchestrator learns to decompose a request, run independent work concurrently, and integrate what the workers return. Experiments on five agentic benchmarks show that our trained orchestrator, MAW-8B, improves its backbone by 8.9 points on average. Further ablations on the challenging DeepPlanning benchmark show that MAW-8B can surpass Claude Sonnet 4.6 as an orchestrator. [1]

1 INTRODUCTION

Large language model (LLM) agents now carry out workflows that combine information gathering, tool use, and multi-step reasoning (Yao et al., 2023; Schick et al., 2023; Qin et al., 2024). Such a workflow usually contains complementary subtasks whose outputs must be assembled into one solution, and multi-agent systems assign each subtask to a separate agent, as exemplified by AutoGen (Wu et al., 2023) and MetaGPT (Hong et al., 2024). Recent systems further execute independent subtasks concurrently, which lets a long workflow finish in fewer sequential rounds (Hadfield et al., 2025; Team et al., 2026). However, distributing a task across agents does not by itself produce a better solution. The benefit instead depends on three decisions, namely how the request is divided, when each part is executed, and how the returned results are combined. In a multi-agent system, one component makes all three, the orchestrator.

Building an effective multi-agent system is difficult, as these three decisions constrain one another. An ideal orchestrator first partitions a request into subtasks that one worker can finish alone. Subtasks that are too fine raise the coordination overhead, while subtasks that are too coarse reduce the team to a single agent. The orchestrator then separates the subtasks that consume the output of another subtask from the subtasks that are genuinely independent. For instance, a request that compares several vendors can retrieve each vendor in parallel, while the comparison itself must wait for every retrieval

[1] Code are available in supplementary materials.

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

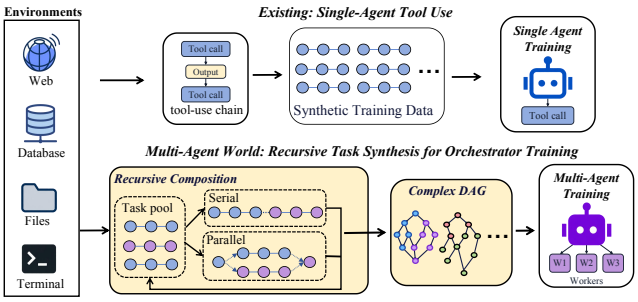


Figure 1: Recursive environment improvement in MAW: serial and parallel composition iteratively transforms seed tasks into complex task graphs for multi-agent orchestration training.

to return. While treating a dependent subtask as independent lets a worker execute on incomplete information, treating an independent subtask as dependent serializes work that could have progressed concurrently. Throughout, the orchestrator never accesses a tool or a record itself. A single agent inspects one observation and corrects its next call. In contrast, the orchestrator observes only what workers report and therefore commits to a decomposition before any evidence exists. Therefore, an orchestrator learns this capability from multi-agent interaction and not from a collaboration procedure fixed in advance.

In most multi-agent systems, orchestration is still designed by hand, as in AutoGen and MetaGPT, where roles, routes, and execution order are fixed in a predefined workflow and the model operates within a collaboration structure that it does not determine. A recent line of work does train orchestration with reinforcement learning (Team et al., 2026; Yuan et al., 2026; Dang et al., 2026), which establishes that the capability is learnable. Scaling that training is limited by the environments available for learning. Specifically, a multi-agent environment offers a task whose subtasks can be distributed across agents, together with a verifier reporting how the team distributed them. Current environment synthesis provides neither property, as AWM (Wang et al., 2026), EnvScaler (Song et al., 2026), Agent-World (Dong et al., 2026) and ToolVerse (Zhou et al., 2026) build code-backed environments for a single agent that observes every tool return itself. A task synthesized this way is usually one chain of calls in which every step consumes the previous observation. In essence, single-agent environments and multi-agent environments differ in two ways. First, a single-agent task carries no controlled branch structure, thereby leaving an orchestrator with no scheduling decision to practice. Second, a single-agent task is scored only by terminal success. Under such a score, a team that partitions the work correctly and misses one integration step is indistinguishable from a team that never decomposes the request.

To this end, we introduce *Multi-Agent World* (MAW), an end-to-end framework for developing multi-agent orchestration. As shown in Figure 1, MAW couples multi-agent environment synthesis with reinforcement learning inside the synthesized environments. Specifically, to close the structural gap, MAW synthesizes multi-agent environments by *recursive environment improvement*. A serial operator links a subtask to the effect it consumes, and a parallel operator places independent subtasks on separate branches. Applying the two operators recursively grows a reference task graph, which semantic reconstruction then rewrites into one coherent user request. After the reference executes, replays from a fresh state, and passes semantic review, MAW admits the task into MAW-Data, our synthesized training set. To close the supervision gap, MAW trains the orchestrator with *schedule-aware process rewards*. These rewards read the recorded execution of the whole team and score tool progress, answer coverage, evidence delivery, and the quality of the schedule itself.

We evaluate MAW-8B, the Qwen3-8B-based orchestrator trained by MAW, on five agentic benchmarks covering multi-turn tool use, interactive task completion, real-world tool execution, long-horizon planning, and broad information seeking. Experimental results show that MAW-8B improves the Qwen3-8B backbone by 8.9 points on average across the five benchmarks, with the largest gains on τ^2-Bench and DeepPlanning. The trained 8B model can also surpass Claude Sonnet 4.6 as an orchestrator on DeepPlanning-Travel: with the same Claude workers, MAW-8B reaches a composite score of 48.90 against Claude's 45.96. More importantly, the ablations show that the two components act together, as composed tasks teach the orchestrator to decompose a request into a team and the schedule-aware reward teaches it to run that decomposition concurrently.

Our contributions are three-fold.

- **End-to-End Multi-Agent Framework**: To the best of our knowledge, MAW is the first framework that develops end to end multi-agent framework, synthesizing multi-agent environments and training inside them, where the task construction record also serves as the signal of its reward.
- **Recursive Environment Improvement and Schedule-Aware Rewards**: We propose recursive environment improvement, which converts single-agent environments into multi-agent environments by treating branch structure as a controlled construction variable, and schedule-aware process rewards, which make the quality of a decomposition visible in the gradient.
- **Strong Performance**: Experimental results on five agentic benchmarks show that MAW-8B improves its 8B backbone by 8.9 points on average, and behavior audits show that the trained orchestrator launches 95% of its assignments concurrently in fewer delegation rounds, which is the behavior that the schedule term rewards.

2 Related Work

Environment Synthesis for Single-Agent Training. Training an agent with reinforcement learning requires environments that execute, hold state, and verify outcomes automatically, and a recent line of work builds such environments at scale for one agent. AWM synthesizes code-backed worlds with structured databases and rule-based checkers, thereby delivering a task with executable tools and a verifiable answer (Wang et al., 2026). EnvScaler separates the construction of an environment skeleton from the generation of scenarios and validators, which raises the yield of usable tasks (Song et al., 2026). Agent-World pairs environment discovery with a diagnostic loop that keeps tasks near the current ability of the policy (Dong et al., 2026), and ToolVerse follows tool dependencies to build long-horizon tasks (Zhou et al., 2026). Further work adapts existing environments through interface components or learns an environment-design role through self-play (Huang et al., 2026; Liu et al., 2026). Overall, these systems produce executable and automatically verified tasks for a single agent that observes every tool return itself. Such a task rarely contains subtasks that several agents can execute simultaneously, and its verifier reports only whether one agent finished.

Multi-Agent Collaboration and Orchestration Training. Multi-agent systems differ mainly in how the collaboration structure is decided. AutoGen organizes collaboration as conversations between agents (Wu et al., 2023), and MetaGPT encodes standardized operating procedures as role-based workflows (Hong et al., 2024). In both cases a human authors the structure and the structure stays fixed during execution. A second group searches the structure automatically, through optimizable agent graphs (Zhuge et al., 2024), workflow generation (Zhang et al., 2025b; Hu et al., 2025), and agentic supernets that select a topology per query (Zhang et al., 2025a). Although search improves the structure, the selected structure still fixes collaboration once execution starts. A third group updates the model that performs the orchestration. Representative methods include evolving orchestration for agent collaboration (Dang et al., 2026), parallel agent reinforcement learning that credits critical steps (Team et al., 2026), and unified agent and tool orchestration with parallel subtask decomposition (Yuan et al., 2026). Overall, learned orchestration is established as feasible. In these studies, the branch structure of the training distribution is inherited from existing benchmarks, thereby remaining neither controlled during construction nor available during reward computation.

Positioning of MAW in Multi-Agent Training. In this work, we focus on how the multi-agent environments an orchestrator trains in relate to the signal it trains under. Unlike environment synthesis that targets a single agent, MAW treats branch structure as a construction variable and composes reference task graphs with serial and parallel operators. Unlike orchestration learning that consumes a fixed task distribution and scores a rollout by terminal success, MAW adopts a more deliberate design by carrying the construction record into training. Specifically, the schedule of the agent team is scored against the independent subtasks that composition has already verified.

3 Multi-Agent World

As shown in Figure 2, MAW develops multi-agent orchestration in two stages. The first stage turns existing single-agent seed pool tasks into multi-agent environments, in which a task carries subtasks

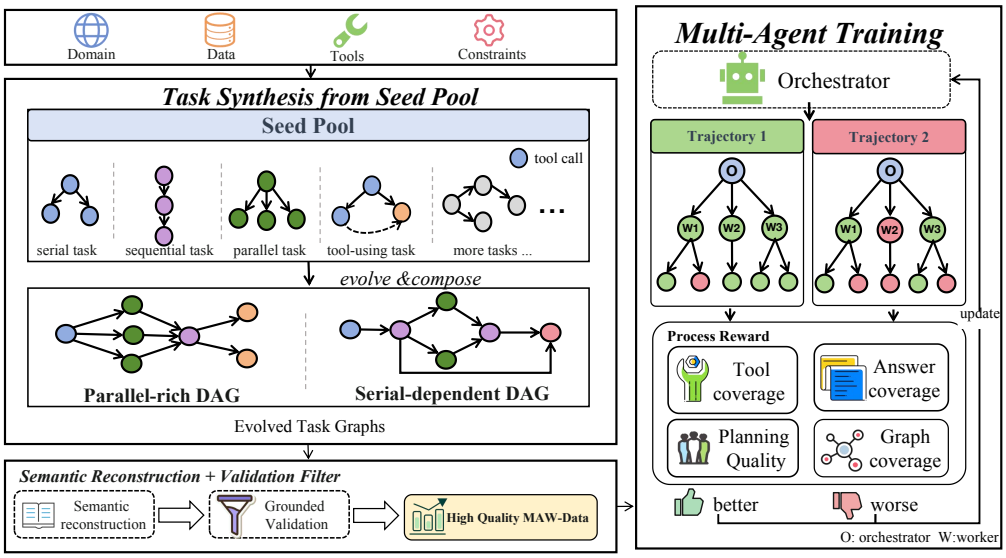


Figure 2: Overview of MAW: compose, reconstruct, and verify multi-agent tasks, then train the orchestrator with schedule-aware process rewards.

that several agents can execute simultaneously. The second stage trains an orchestrator inside those environments under a reward that reads how the agent team actually distributed the work.

3.1 From Single-Agent to Multi-Agent Environments

An agentic environment is written as $\mathcal{E} = (\mathcal{S}, \mathcal{U}, f_{\text{exec}}, s_0)$, where $\mathcal{S}$ is a database-backed state space, $\mathcal{U}$ is the tool interface, and s_0 is the initial state. The transition $f_{\text{exec}}(s, u) = (s', o)$ executes a tool call u and returns an observation o. Following the synthesis setting of AWM and Agent-World (Wang et al., 2026; Dong et al., 2026), a task in $\mathcal{E}$ is written as $x = (q, G, a^\star, \Delta^\star)$. Here q is the user request, $a^\star$ is the reference answer, and $\Delta^\star$ is the required change of the database. The reference task graph $G = (V, A)$ has one node $v \in V$ per reference tool call, and an edge $(u, v) \in A$ means that v consumes an output or a state effect of u.

A multi-agent environment extends this setting with a delegation interface and an independence record. The interface $\mathcal{H}$ lets an orchestrator create workers and assign subtasks to them, which is what makes a team executable at all. Task composition creates independent subtasks within a request. The record Π identifies verified pairs of independent tool calls, allowing the process reward to assess whether the orchestrator assigns them to different workers for concurrent execution. We write a multi-agent environment as $\mathcal{E}^{\text{ma}} = (\mathcal{S}, \mathcal{U}, f_{\text{exec}}, s_0, \mathcal{H})$ and a multi-agent task as $x^{\text{ma}} = (q, G, a^\star, \Delta^\star, \Pi)$. Here each pair $(u, v) \in \Pi$ consists of two reference tool calls verified to be executable in either order without changing the final answer or database state. Orchestration training consumes this record, without which a verifier reports only whether the team produced the correct answer and leaves the correctness of a simultaneous execution unmeasured. Existing pipelines produce $(q, G, a^\star, \Delta^\star)$ for one agent and leave Π empty, which is why their tasks train tool use and not orchestration.

3.2 Recursive Environment Improvement

In this stage, we aim to turn existing single-agent backends into multi-agent environments without building new ones from scratch, which is challenging as branch structure cannot be requested from a generator directly. When a model is asked for a parallel task, it tends to produce two unrelated questions in one prompt, leaving an executable reference that is either trivial or inconsistent. MAW instead constructs branch structure at the level of reference graphs, where independence is a checkable property, and writes the graph as natural language only afterwards.

Recursive Composition. Before any natural language is generated, composition introduces a controlled amount of dependency and independence into a task. From a seed pool $\mathcal{C}$ of single-agent

tasks in the same backend, MAW composes two of them into a candidate graph,

$$\tilde{G} = f^{z}_{\mathrm{comp}}(G_1, G_2), \qquad z \in \{\mathrm{serial}, \mathrm{parallel}\}, \tag{1}$$

where the serial operator adds an edge between a pair $(v_1, v_2) \in V_1 \times V_2$ whose output or state effect fills an argument of the other. In contrast, the parallel operator takes the disjoint union with no edge between the two sides and lets one answer node consume both. After composition, a candidate graph re-enters $\mathcal{C}$ and acquires further subtasks in later rounds, so mixed structure arises from repeated application. Appendix B gives a worked example.

Semantic Reconstruction and Verification. A composed graph is not yet a request a user would write, as the two parents may refer to different entities or reporting formats. Reconstruction produces one coherent request and an executable reference plan, written as $(q, G) = f_{\mathrm{recon}}(\tilde{G}, q_1, q_2, \mathcal{U})$. In this stage, MAW aligns entities across the two sides, merges nodes that return the same content, and inserts a lookup when the combined goal needs one. The request states the desired outcome and exposes neither the reference plan without any hint about answers or how many workers to use.

After reconstruction, MAW checks each composed task for redundant calls and requirements that cannot be jointly satisfied. Executing the plan from a reset state gives $(a^\star, \Delta^\star) = f^{*}_{\mathrm{exec}}(G, s_0)$, and MAW then replays it from a fresh state. MAW accepts a task only when execution, replay, and an independent review all pass, written as $f_{\mathrm{gate}} = f_{\mathrm{run}} \cdot f_{\mathrm{replay}} \cdot f_{\mathrm{sem}} \in \{0, 1\}$ and specified in Appendix A. Admission is the step that keeps a composed candidate from entering training as an unsolvable or self-contradictory task.

Although composition proposes that two subtasks lie on separate branches, the proposal is not yet verified, as reconstruction may introduce a shared entity that makes one side depend on the other. MAW verifies every proposed branch by execution before recording it,

$$\Pi = \big\{(v, v') \in V_{\mathrm{r}} \times V_{\mathrm{r}} \,:\, v \not\prec v',\ v' \not\prec v,\ f_{\mathrm{perm}}(v, v') = 1\big\}, \tag{2}$$

where $V_{\mathrm{r}} \subseteq V$ is the set of read-only calls and $v \prec v'$ denotes that v' is reachable from v in G. The test $f_{\mathrm{perm}}(v, v')$ executes the reference plan again with the order of the two calls exchanged, returning one when the answer and the database effect are unchanged. Together with its resettable environment, its checker, and its hidden reference, this record converts a synthesized task into a multi-agent task. One construction therefore produces two things at once, namely the branches an orchestrator can act on and the ground truth against which its use of those branches is scored. We collect these verified tasks in MAW-Data. Section 3.3 consumes the second of them.

3.3 Multi-Agent Training with Schedule-aware Process Rewards

In this stage, we train the orchestrator on the synthesized multi-agent tasks. As task difficulty increases, all rollouts sampled for a task often fail to complete it, receiving zero outcome reward even when some make useful progress. These identical rewards yield zero group-relative advantages, making it difficult to learn from outcome supervision alone. MAW therefore introduces a schedule-aware process reward that evaluates partial progress and scheduling decisions from the team's execution trace, providing a learning signal before the policy can reliably complete the task.

Team-Level Outcome Verification. In a multi-agent setting, the outcome term determines whether the team completed the task by verifying the environment and not the report of any agent. For a trajectory τ with final answer a_τ and final state s_τ, the outcome term requires both an accepted answer and an accepted database state,

$$f_{\mathrm{out}}(\tau) = V^{\mathrm{ans}}\big(a_\tau, a^\star\big) \cdot V^{\mathrm{state}}\big(\Delta_\tau, \Delta^\star\big) \in \{0, 1\}, \tag{3}$$

where V^{ans} and V^{state} are the answer and state checkers of the task, each returning one when the submitted result is accepted and zero when an episode ends without a submission. The state checker compares the required database mutations and rejects unexpected changes. Checking Δ_τ verifies that the requested changes were actually made, rather than relying on the agents' claims of completion.

Schedule-aware Process Reward. Process rewards assess intermediate steps and can improve credit assignment in sparse-reward settings (Ng et al., 1999; Lightman et al., 2024; Uesato et al., 2022). The process term scores four properties of a team execution, namely the reference work

completed, the requested answer present, the evidence returned by workers, and the independent work scheduled as independent,

$$f_{\text{proc}}(\tau) = \alpha_{\text{t}} f_{\text{tool}}(\tau) + \alpha_{\text{a}} f_{\text{ans}}(\tau) + \alpha_{\text{d}} f_{\text{del}}(\tau) + \alpha_{\text{s}} f_{\text{sch}}(\tau), \tag{4}$$

where every term lies in $[0, 1]$ (Table 9). The tool term f_{tool} is the fraction of reference calls in V matched once by a successful event with equivalent arguments and observation. The answer term f_{ans} gives recall-oriented partial credit over the top-level fields of the answer. The delivery term f_{del} is the fraction of assigned subtasks whose worker return carries the required evidence, which measures the handoff a single agent never performs. The schedule term is the part of the reward that no terminal verifier can express, and it reads the independence record produced by Eq. 2,

$$f_{\text{sch}}(\tau) = (1 - \delta_\tau) \cdot f_{\text{int}}(\tau) \cdot \frac{|\Pi_\tau|}{|\Pi|}, \tag{5}$$

The leading ratio is the fraction of verified independent pairs that the team actually ran as independent, where $\Pi_\tau \subseteq \Pi$ contains the pairs assigned to two different workers under assignment intervals that do not depend on each other. Two factors keep that ratio honest, since δ_τ discounts assignments that repeat a covered subtask and f_{int} discounts branch results that never reach the final answer. Eq. 5 is where synthesis and learning meet, as Π exists only because composition proposed a branch and a reordered execution confirmed it. A team is therefore credited for parallel work only on branches the environment has proven parallel, which is a judgement that terminal success cannot make.

Orchestrator-only Policy Optimization. Partial progress requires enough weight to distinguish failing rollouts and little enough to never compete with a verified success. The two terms are combined as

$$R(\tau) = f_{\text{out}}(\tau) + \lambda\, f_{\text{proc}}(\tau), \qquad \lambda = 0.3, \tag{6}$$

which gives a successful team at least 1 and an unsuccessful team at most λ. Process credit thereby separates failed rollouts without ever outweighing success. We optimize only the orchestrator π_θ, which delegates tool execution to frozen workers π_ϕ through the interface described in Appendix C. For each task we sample K team trajectories from independently reset environments and compute the group-relative advantage $A_i = (R_i - \bar{R})/(\text{std}(R_1, \ldots, R_K) + \epsilon)$ of the i-th trajectory. The orchestrator is then updated with the clipped GRPO objective (Shao et al., 2024), written as

$$J(\theta) = \mathbb{E}\Big[\tfrac{1}{K}\textstyle\sum_{i=1}^{K} \tfrac{1}{|\mathcal{T}_i|} \sum_{t \in \mathcal{T}_i} \min\big(\rho_{it}(\theta) A_i,\ \text{clip}(\rho_{it}(\theta), 1 - \epsilon_l, 1 + \epsilon_h) A_i\big)\Big], \tag{7}$$

where $\mathcal{T}_i$ contains only the tokens generated by the orchestrator and ρ_{it} is the ratio between the current and the behavior policy on that token. Worker generations and tool observations enter the context of the orchestrator and stay out of $\mathcal{T}_i$. Since π_ϕ is never updated, every measured change comes from orchestration and not from worker quality.

4 Experiments

4.1 Setup

Benchmarks. We evaluate on five agentic benchmarks: BFCL Multi-Turn (Patil et al., 2025), τ^2-Bench (Barres et al., 2025), and MCPMark (Wu et al., 2026) for *tool use*; DeepPlanning (Zhang et al., 2026) and WideSearch (Wong et al., 2026) for *long-horizon tasks*. The two groups test dependent tool execution and the coordination of planning and information gathering, respectively. Appendix D.1 details the tasks, evaluation splits, metrics, and protocols.

Baselines. We compare MAW-8B with three groups of systems. *Frontier models* indicate the headroom of each benchmark: GPT-5.6 Terra is the reference for the tool-use group, while GPT-5.6 Terra and Claude Sonnet 4.6 are evaluated as single agents on the long-horizon group (Appendix D.1). *Open-source backbones* are the untrained Qwen3-8B and Qwen3-14B models (Yang et al., 2025), of which Qwen3-8B initializes every trained system.

Single-agent environment synthesis covers AWM (Wang et al., 2026) and EnvScaler (Song et al., 2026). For both baselines, we download the checkpoints released by the respective authors and evaluate them in our local evaluation environment.

Table 1: Tool-use results (%): BFCL accuracy, τ^2-Bench task success, and MCPMark filesystem pass@1. Avg is the unweighted mean within each benchmark; **bold** marks the best 8B result.

Model	BFCL Multi-Turn					τ^2-Bench				MCPMark
	Base	MissFunc	MissParam	LongCtx	Avg	Airline	Retail	Telecom	Avg	FS
Frontier Models										
GPT-5.6 Terra	64.50	57.50	49.50	35.00	51.63	64.00	64.91	50.00	59.64	55.00
Open-Source Backbones										
Qwen3-8B	32.00	33.50	22.00	28.00	28.88	22.00	43.20	18.40	27.87	12.50
Qwen3-14B	52.00	37.00	41.00	41.50	42.88	46.00	51.75	29.82	42.53	15.00
Single-Agent Environment Synthesis										
AWM-8B	23.50	9.00	20.50	18.00	17.75	18.00	35.96	27.19	27.05	10.00
EnvScaler-8B	**48.50**	26.00	30.50	**38.50**	35.88	14.00	34.21	20.18	22.80	**15.00**
MAW-8B (ours)	45.00	**37.00**	**33.50**	33.00	**37.13**	**38.00**	**51.75**	**31.58**	**40.44**	**15.00**

Implementation. We build the orchestrator and the workers on the same Qwen3-8B base model. We first train the worker on 10,860 seed-pool tasks. MAW-Data contains 2,581 admitted tasks across 539 environments. With the worker policy frozen, we train the orchestrator on MAW-Data with GRPO under Eq. 6 and sample $K = 8$ team trajectories per task from independently reset environments. All training and evaluation run on 8*NVIDIA H20 GPUs.

4.2 Results on Tool-use Benchmarks

Table 1 reports the tool-use group. MAW-8B improves both multi-turn function calling and interactive task completion. On BFCL Multi-Turn, it reaches 37.13 on average, improving the backbone by 8.25 points and EnvScaler-8B, the strongest single-agent synthesis baseline, by 1.25 points. It also achieves best performance on the missing-function and missing-parameter settings. On τ^2-Bench, MAW-8B reaches 40.44 and is the best 8B system in every domain, improving the backbone by 12.57 points and the strongest synthesis baseline by 13.39 points. It exceeds the untrained Qwen3-14B on telecom and matches it on retail. On MCPMark, MAW-8B solves 6 of 40 filesystem tasks for a pass@1 of 15.00, improving the backbone by 2.50 points, tying EnvScaler for the best 8B result, and matching Qwen3-14B. The gain is smaller than on BFCL and τ^2-Bench, and all 8B systems score between 10.00 and 15.00.

The gain concentrates on the settings where a team can verify before committing, as a withheld function, a missing parameter, or a policy constraint of the simulated user is recovered by assigning a verification subtask.

4.3 Results on Long-horizon Benchmarks

Table 2 reports the long-horizon group, where every benchmark leaves a large gap between the backbone and the frontier models. On DeepPlanning, MAW-8B raises the travel composite score from 7.73 to 19.84 and the shopping match rate from 24.73 to 47.07, exceeding the untrained Qwen3-14B on both. On WideSearch, it raises item F1 from 20.51 to 24.40. In contrast, the single-agent synthesis baselines gain little or degrade on this group, as both fall below the backbone on travel planning and on WideSearch, and only AWM gains on shopping.

Training on longer single-agent call chains thus does not transfer to tasks whose evidence must be gathered along several branches. Recursive environment improvement produces that branch structure during synthesis. The resulting tasks train the orchestrator to assign independent retrievals and constraint checks to separate workers and to collect their results.

The gains also vary with the structure of the evaluation task. The gain is largest on shopping, where the products of a list can be searched independently and combined under one budget, and smallest on WideSearch, where a table row still requires several dependent lookups. The finer scores in Table 2 locate the travel gain in the commonsense score, which MAW-8B improves by 24.64 points, while

Table 2: Results on long-horizon benchmarks (%). Travel: commonsense (CS), personalized (PS), and composite (Comp) scores averaged over Chinese and English. Shopping: product match rates with a product-count-weighted Avg. WideSearch: row/item F1 on 100 English tasks. Bold marks the best 8B result.

Model	DeepPlanning-Travel			DeepPlanning-Shopping				WideSearch	
	CS	PS	Comp	L1	L2	L3	Avg	Row-F1	Item-F1
Frontier Models									
GPT-5.6 Terra	88.54	51.25	69.90	89.30	77.83	76.24	82.05	52.37	75.23
Claude Sonnet 4.6	54.64	35.00	44.82	80.47	67.39	74.26	73.81	25.27	51.38
Open-Source Backbones									
Qwen3-8B	15.05	**0.42**	7.73	32.09	17.39	25.74	24.73	1.92	20.51
Qwen3-14B	23.07	3.33	13.20	46.98	37.39	35.64	40.84	2.46	17.66
Single-Agent Environment Synthesis									
AWM-8B	8.49	**0.42**	4.45	27.91	24.35	24.75	25.82	1.89	19.50
EnvScaler-8B	5.31	0.00	2.66	25.58	17.39	20.79	21.25	1.01	16.95
MAW-8B (ours)	**39.69**	0.00	**19.84**	**52.56**	**45.65**	**38.61**	**47.07**	**2.88**	**24.40**

Table 3: Orchestrator and worker ablations on DeepPlanning-Travel (%). SA-8B: single-agent initialization. Time and team-wide tokens are per-episode. **Bold** marks the best score.

Orchestrator	Workers	CS ↑	PS ↑	Comp ↑	Time (s) ↓	Tokens (k) ↓
–	SA-8B (single agent)	30.05	2.92	16.48	36.0	$\geq$ 128.8
–	Claude (single agent)	54.64	35.00	44.82	92.9	$\geq$ 330.4
Claude	SA-8B	31.62	10.16	20.89	421.6	$\geq$ 637.6
MAW-8B	SA-8B	39.69	0.00	19.84	135.1	$\geq$ 161.7
Claude	Claude	50.20	**41.71**	45.96	469.0	$\geq$ 184.3
MAW-8B	Claude	**58.22**	39.57	**48.90**	108.9	$\geq$ 200.9

the personalized score, which is all-or-nothing over the user constraints of a case, stays at or near zero for every 8B system.

Overall, the results on this group highlight the transfer of synthesized branch structure to long-horizon tasks that the training distribution never contained, and Section 5.1 separates the contribution of the orchestrator from that of the workers. Across the two groups, MAW-8B improves the backbone by 8.9 points on average, using the Avg columns for BFCL and τ^2-Bench, filesystem pass@1 for MCPMark, the mean of travel composite and shopping match rate for DeepPlanning, and item-level F1 for WideSearch.

5 Analyses

5.1 Effect of the Orchestrator and the Workers

To separate the contribution of the trained orchestrator from that of the workers it drives, we swap each role on DeepPlanning-Travel while holding the other fixed. We report the commonsense, personalized, and composite scores together with the wall-clock time and the token usage of an episode. SA-8B, the single-agent checkpoint trained on the seed pool, and Claude Sonnet 4.6 each solve the task alone as reference points. The four multi-agent settings pair either MAW-8B or Claude as orchestrator with either SA-8B or Claude as workers.

With worker capability fixed, the trained orchestrator improves task success. As shown in Table 3, it raises the composite score from the SA-8B reference of 16.48 to 19.84. With SA-8B workers, MAW-8B scores 19.84 against the Claude orchestrator's 20.89, a gap of 1.05 points. It leads Claude on commonsense by 8.07 points but trails on personalization by 10.16 points, locating the remaining weakness in the user constraints that the personalized score checks all-or-nothing.

Table 4: Training data and reward ablations on DeepPlanning (%) with the same multi-agent architecture. O: outcome reward; P: schedule-aware process reward. Travel metrics follow Table 2; Shopping pools product matches across L1–L3.

Setting	Training data	Reward	DeepPlanning-Travel			Shopping
			CS ↑	PS ↑	Comp ↑	Match ↑
MAW (ours)	MAW-Data	$O + P$	39.69	0.00	19.84	47.07
w/o process reward	MAW-Data	O	24.27	0.00	12.14 (-7.70)	46.89 (-0.18)
w/o recursive environment improvement	Seed pool	O	22.03	2.92	12.47 (-7.37)	32.05 (-15.02)

Stronger workers provide a further gain without retraining the orchestrator. Replacing SA-8B workers with Claude raises the composite score from 19.84 to 48.90. With the same Claude workers, MAW-8B also outperforms the Claude orchestrator by 2.94 points, 48.90 against 45.96, with higher commonsense but slightly lower personalized scores. It exceeds Claude acting alone by 4.08 points, whereas the all-Claude team scores 1.14 points below that single-agent reference. These results show that the learned orchestration transfers across workers and that both worker capability and the choice of orchestrator affect task success.

5.2 Effect of the Training Data and the Reward

To verify that both components of MAW contribute, we ablate them in sequence while keeping others fixed. The second setting removes the schedule-aware process reward and trains on the same MAW-Data tasks with the outcome reward only. The third setting further replaces MAW-Data with the seed pool, whose tasks carry simple tool use problem. Table 4 reports the travel scores and the shopping match rate pooled over the three difficulty levels.

On travel, the process reward is necessary to obtain the full gain from MAW-Data. As shown in Table 4, removing it lowers the composite score from 19.84 to 12.14, which is no better than training on the seed pool with the outcome term alone. Under a outcome-only reward, a group of team often rollouts that all fail produces identical returns, so MAW-Data provides branch structure that the policy is never rewarded for using. On shopping, the training data has the larger effect in these comparisons. The match rate is nearly unchanged without the process reward, at 46.89 against 47.07, and falls by 14.84 points when MAW-Data is further replaced with the seed pool. A product list decomposes into independent searches whose outcome the terminal verifier already separates.

The two components therefore act on different tasks, where MAW-Data supplies independent subtasks to schedule and the process reward converts them into a gain on tasks whose rollouts rarely succeed early in training. Overall, training on MAW-Data alone does not reach the full system on travel, and the process reward is the larger factor on the benchmark with the sparsest outcome signal.

Further Analyses. Appendix A.2 profiles MAW-Data statistics , Appendix B traces a synthesized tasks from their parents to their independence records, and Appendix E measures how three orchestrators of Table 4 distribute work on DeepPlanning-Travel.

6 Conclusion

We presented MAW, an end-to-end framework that develops multi-agent orchestration by synthesizing multi-agent environments and training an orchestrator inside them. Recursive environment improvement composes the reference graphs of single-agent tasks, reconstructs each composed graph into one coherent request, and admits it only after execution, replay, and semantic review. During training, MAW uses a schedule-aware process reward to develop the orchestrator's ability to decompose tasks, schedule independent subtasks concurrently, and integrate worker results. Experiments on five agentic benchmarks show that this design improves task success on tool-use and long-horizon tasks. On DeepPlanning-Travel, the trained MAW-8B surpasses Claude as an orchestrator when both use the same Claude workers. This result shows that training specifically for orchestration can enable an 8B model to coordinate a given team more effectively than a frontier model, while the worker ablations show that the advantage depends on the team's execution capability.

AI USE STATEMENT

In this work, we used generative AI tools for synthetic dataset generation and manuscript polishing. Specifically, Claude Sonnet 4.6 served as the task generator and semantic reviewer in the MAW-Data synthesis pipeline, including reconstructing composed task graphs into coherent requests and reviewing candidate tasks against predefined criteria, as described in Section 3.2 and Appendix A. Generated tasks were admitted only after passing programmatic checks, execution and fresh-state replay, and semantic review. We also used generative AI tools for final-stage language polishing and grammar refinement of the manuscript. The authors reviewed all AI-assisted manuscript revisions for accuracy and consistency with the intended content and take responsibility for the final paper and the generated dataset.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure the reproducibility of this work by providing detailed descriptions of the proposed method, experimental settings, implementation details, and evaluation protocols in the main paper and supplementary materials. The supplementary materials include the source code required to reproduce our experiments, and we will publicly release the code after the review process to further facilitate verification and future research. The content of this paper can be completely reproduced.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. tau^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, et al. Multi-agent collaboration via evolving orchestration. *Advances in neural information processing systems*, 38:165025–165059, 2026.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, et al. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026. URL https://arxiv.org/abs/2604.18292.

Jeremy Hadfield, Barry Zhang, Kenneth Lien, Florian Scholz, Jeremy Fox, and Daniel Ford. How we built our multi-agent research system. Anthropic Engineering Blog, June 2025. URL https://www.anthropic.com/engineering/multi-agent-research-system.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Chengsong Huang, Zifeng Wang, Rujun Han, Jun Yan, Yanfei Chen, Zoey CuiZhu, Ke Jiang, Peng Xia, Han Yu, Yufan Zhuang, et al. Envharness: Awakening static worlds for agent learning, 2026. URL https://arxiv.org/abs/2608.19880.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Bo Liu, Simon Yu, Yiding Jiang, Ao Qu, Andrew Zhao, Zichen Liu, Junsu Kim, Zijian Zhou, Seungone Kim, Tongzheng Ren, et al. Spade: Self-play in adaptive synthetic executable environments, 2026. URL https://arxiv.org/abs/2608.19197.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 278–287. Morgan Kaufmann, 1999. URL https://people.eecs.berkeley.edu/~russell/papers/icml99-shaping.pdf.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=2GmDdhBdDk.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in neural information processing systems*, 2023. URL https://openreview.net/forum?id=Yacmpz84TH.

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

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, 2026.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Ziwei Chai, Y Charles, HS Che, Cheng Chen, et al. Kimi k2. 5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2602.10090.

Ryan Wong, Jiawei Wang, Li Chen, Yan Gao, Xuan Zhou, Zuo Wang, Kai Xiang, Ge Zhang, Wenhao Huang, Yang Wang, et al. Widesearch: Benchmarking agentic broad info-seeking. In *International Conference on Learning Representations*, volume 2026, pp. 10012–10086, 2026.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Zijian Wu, Xiangyan Liu, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, Zirui Wang, et al. Mcpmark: A benchmark for stress-testing realistic and comprehensive mcp use. In *International Conference on Learning Representations*, volume 2026, pp. 79852–79895, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Wenzhen Yuan, Wutao Xiong, Fanchen Yu, Shengji Tang, Ting Liu, Tao Chen, Peng Ye, Yuzhuo Fu, Wanli Ouyang, and Lei Bai. Small model as master orchestrator: Learning unified agent-tool orchestration with parallel subtask decomposition, 2026. URL https://arxiv.org/abs/2604.17009.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet, 2025a. URL https://arxiv.org/abs/2502.04180.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025b.

Yinger Zhang, Shutong Jiang, Renhao Li, Jianhong Tu, Yang Su, Lianghao Deng, Xudong Guo, Chenxu Lv, and Junyang Lin. Deepplanning: Benchmarking long-horizon agentic planning with verifiable constraints. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7377–7407, 2026.

Shuaiyu Zhou, Fengpeng Yue, Zengjie Hu, Yuanzhe Shen, Chenyang Zhang, Cao Liu, Ke Zeng, et al. Toolverse: Unlocking massive environments and long-horizon tasks for agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2607.15660.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs, 2024. URL https://arxiv.org/abs/2402.16823.

A MULTI-AGENT ENVIRONMENT SYNTHESIS

A.1 ADMISSION CHECKS

A candidate of Section 3.2 is a reconstructed request q together with its reference plan G. Before a candidate is admitted, MAW runs the five checks of Table 5 in the order listed. The first four checks are executed by code and make up the factors f_{run} and f_{replay} of the admission decision f_{gate}: plan validity and graph consistency inspect the plan before it runs, execution runs it once and tests the checker, and replay runs it a second time. The fifth check is the factor f_{sem}, a review by a language model that reads the request together with the recorded execution.

Table 5: Admission checks in execution order, with typical rejections. Here, $a^\star$ and $\Delta^\star$ denote the reference answer and database state, respectively.

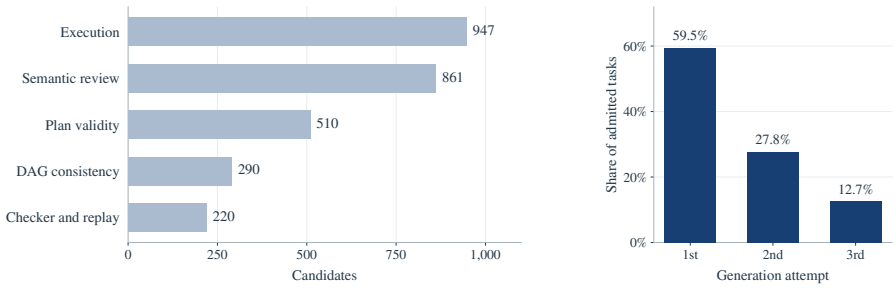
Check	Passes when	Typical rejection
Plan validity	Calls use backend tools, schema-valid arguments, unique identifiers, and references to outputs produced by earlier calls.	A call fills an argument from an output that no earlier call produces.
Graph consistency	G has one node per reference call plus an answer node; edges are acyclic, forward, and execution-consistent, with state-order edges supported by relevant recorded changes.	The edges contain a cycle, or a state-order edge points from a write that changed nothing the later call reads.
Execution	Execution from a reset state completes with $(a^\star, \Delta^\star)$; the checker accepts both and rejects deliberately corrupted copies of each.	A tool call raises an error, or the checker accepts a corrupted answer.
Replay	A fresh-state replay reproduces the answer and database effect, without changing records outside $\Delta^\star$.	The two executions return different answers, or the plan mutates a record that $\Delta^\star$ does not list.
Semantic review	A large language model review of the request, plan, and execution accept all six criteria in Table 6 with evidence.	The request drops a goal of one parent, or a repeated call neither adds information nor observes changed state.

A.2 MAW-DATA PROFILE

Starting from 10,860 seed tasks across 598 environments, the synthesis batch evaluates 5,537 two-parent reference unions after rule-based pruning, ultimately admitting 2,581 tasks across 539 environments into MAW-Data. The generator and the semantic reviewer are Claude Sonnet 4.6. The review runs twice under the six criteria of Table 6, where the second review starts only after the first passes and sees no earlier verdict, and every admitted task passes both.

Table 6: Criteria of the semantic review. Every criterion requires supporting evidence from the recorded execution.

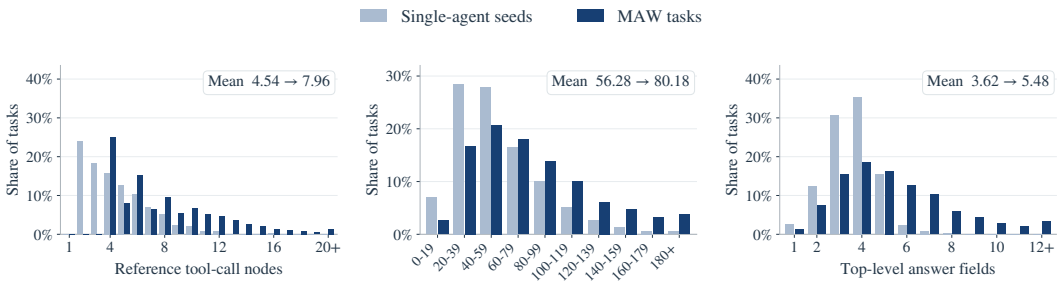
Criterion	Question
Task fulfillment	Does the execution accomplish the generated request?
Answer completeness	Does the answer include every requested result?
Parent preservation	Are the functional outcomes of both parents retained?
Workflow coherence	Do the operations serve one coherent task?
State consistency	Do the state claims agree with the execution and the final database?
Justified repetition	Does each repeated call add information, observe changed state, or serve a distinct purpose?



(a) Final rejection by gate **(b) Attempt of acceptance**

Figure 3: Candidate outcomes and repair. Left, the final disposition of failed candidates, assigned to the last stage it reached. Right, the attempt on which an admitted task passed.

As shown in Figure 3, execution and semantic review account for the most final rejections (947 and 861, respectively). Among admitted tasks, 59.5% pass on the first attempt, while 40.5% require repair, passing on the second (27.8%) or third (12.7%) attempt.



(a) Reference DAG size **(b) Instruction length** **(c) Answer breadth**

Figure 4: Reference structure of the seed pool and of MAW-Data over 10,860 seeds and 2,581 admitted tasks in 539 environments. Composition grows the number of answer fields far more than the number of tool calls, which indicates that an admitted task asks for more separable results and not simply a longer call chain.

As shown in Figure 4, reference tool calls grow by 19.6% relative to the two parents of each admitted task, distinct tools by 3.9%, and top-level answer fields by 57.3%. The matched baseline of a task is the mean of its two parents, a parent reused by several descendants contributes once per descendant, and the answer-field comparison excludes three seeds whose answer is not a dictionary. A composed task thereby asks for more results that can be produced independently, which is the property an orchestrator can exploit.

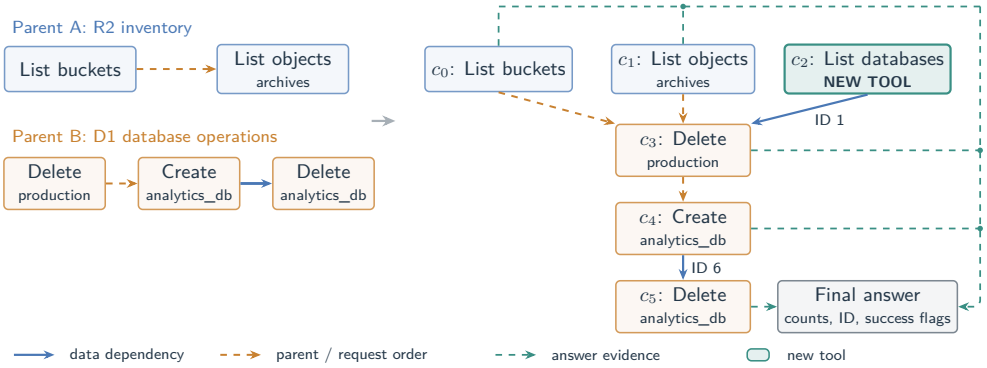
B SYNTHESIS CASE STUDIES

Case 1 Adding database discovery to a composed task

Environment. Cloudflare R2 object storage and D1 database management in the frozen backend.

Lineage. One reference union of two seeds: S0005 counts buckets and objects in `archives`; S0006 deletes `production`, creates `analytics_db`, and deletes the new database. Reconstruction retains their five calls and introduces one tool absent from both parents.

Composed and reconstructed graphs



The new `d1_list_databases` call resolves `production` to ID 1 for deletion. Calls increase from 5 to 6 and distinct tools from 4 to 5. The parent weak edge between the R2 reads is removed because `archives` is supplied in the request; the reconstructed graph records the two ID dependencies shown above.

Generated request

Count the total number of R2 buckets in my Cloudflare account, count how many objects are stored in the R2 bucket named 'archives', then delete the D1 database named 'production', create a new D1 database named 'analytics_db', and delete the newly created 'analytics_db' database. Report the total bucket count, the object count in 'archives', the database ID that was assigned to 'analytics_db' when it was created, and confirm whether all three database operations (delete production, create analytics_db, delete analytics_db) succeeded.

Reference execution

	Tool call	Observation
c_0	`r2_list_buckets()`	6 buckets.
c_1	`r2_list_objects(bucket_name='archives')`	14 objects.
c_2	`d1_list_databases()`	`production` has ID 1.
c_3	`d1_delete_database(database_id=1)`	Success; deleted ID 1.
c_4	`d1_create_database(name='analytics_db')`	Created database ID 6.
c_5	`d1_delete_database(database_id=6)`	Success; deleted ID 6.

Reference answer and record

```
{
  "total_r2_buckets": 6,
  "objects_in_archives": 14,
  "analytics_db_id": 6,
  "delete_production_success": true,
  "create_analytics_db_success": true,
  "delete_analytics_db_success": true
}
```

State. Only `production` is removed from the final state; `analytics_db` is created and deleted. R2 is unchanged.

Independent pairs. (c_0, c_1), (c_0, c_2), (c_1, c_2), confirmed by all six permutations of the three reads.

Ideal schedule

Run $\{c_0, c_1, c_2\}$ in parallel, then c_3, c_4, and c_5 in the requested database-operation order, then integrate the results: four levels of tool calls instead of six. The task passed execution and both semantic reviews on its first attempt.

C Multi-Agent Runtime and Information Boundary

The delegation-only harness separates orchestration from worker execution. As summarized in Table 7, the orchestrator registers workers with specified roles and permitted tool subsets, assigns work to them, and submits the final answer. A single assignment executes synchronously, whereas a batch assignment launches workers concurrently and returns after their results have joined. All workers follow the frozen policy.

Table 7: Orchestration interface. A worker is registered with a role and a permitted tool subset, a single assignment is synchronous, and a batch assignment launches workers concurrently and returns when their results join.

Action	Required information and effect
`create_subagent`	Name, system prompt, and allowed tool names, which registers a worker that follows the frozen policy.
`assign_task`	Worker name and subtask prompt, which executes one assignment and returns its result.
`assign_tasks`	A list of assignments, which permits concurrent worker inference and joins the returns.
`submit_answer`	The final answer in the required format, which triggers completion and verification.

Worker reasoning may overlap across concurrent assignments, while database tool operations commit under a transaction lock. This serializes database commits and gives concurrent execution a well-defined order of state updates. The runtime records each model generation with its token counts and timing, and each tool event with its arguments, observation, status, and owning assignment. These execution records support reward computation during training and the trajectory analysis described in Appendix E during evaluation.

The execution record is distinct from the information exposed to the policy. Table 8 specifies the information available to the orchestrator, each worker, and the reward service during an episode. The orchestrator receives tool and worker returns, while each worker operates within its assigned context and permitted tool subset. The reference plan, expected answer, independence record Π, and reward component scores are accessible only to the reward service. Reward reports are never returned as policy observations, so neither the orchestrator nor the workers can use these signals to guide execution within an episode.

Table 8: Information boundary during an episode. Reward reports are never returned as observations, so the policy cannot read its own process score.

Information	Orchestrator	Worker	Reward service
User request	Yes	Assigned content	Yes
Allowed tool descriptions	Yes	Permitted subset	Yes
Tool and worker returns	Returned content	Own context	Logged content
Reference plan and expected answer	No	No	Yes
Independence record Π	No	No	Yes
Reward component scores	No	No	Yes

D Training and Evaluation Protocol

Table 9 summarizes the training configuration of MAW-8B. The orchestrator and workers are initialized from the same single-agent checkpoint based on Qwen3-8B. The workers is trained on the seed pool data. During orchestrator training, the workers remain frozen, while the orchestrator is optimized on MAW-Data using GRPO with eight rollouts per task and a combination of outcome and process rewards. The process reward coefficient is set to $\lambda = 0.3$.

Table 9: Training configuration of MAW-8B. The orchestrator and the workers start from the same single-agent checkpoint, and only the orchestrator is updated.

Setting	Value
Backbone	Qwen3-8B
Orchestrator initialization	Single-agent checkpoint
Worker initialization and update	Same checkpoint, frozen
Optimization	GRPO with outcome and process reward
Rollouts per task	8
Process coefficient λ	0.3
Process weights $(\alpha_{\mathrm{t}}, \alpha_{\mathrm{a}}, \alpha_{\mathrm{d}}, \alpha_{\mathrm{s}})$	$(0.25, 0.25, 0.20, 0.30)$
Cost in the training reward	None
Completion protocol	At least one delegation

D.1 Benchmark Protocol

Tool-use benchmarks. BFCL Multi-Turn (Patil et al., 2025) evaluates multi-turn function calling in the base setting and under missing functions, missing parameters, and long contexts. τ^2-Bench (Barres et al., 2025) measures interactive task completion with a simulated user in the airline, retail, and telecom domains. MCPMark (Wu et al., 2026) evaluates stateful task completion on real Model Context Protocol services; we use its filesystem tasks and report pass@1. Together, these benchmarks test whether a team can preserve correctness across dependent tool calls and interactions.

Long-horizon benchmarks. DeepPlanning (Zhang et al., 2026) covers multi-day travel planning and multi-product shopping. Travel is evaluated by a commonsense score over feasibility constraints, a personalized score over user constraints, and their composite. Shopping is evaluated by product match rate at three difficulty levels. WideSearch (Wong et al., 2026) requires broad information gathering to complete a structured table and reports row-level and item-level F1. These tasks require gathering evidence and integrating it under global constraints.

We report results by setting, domain, or difficulty level where applicable. Benchmark-level averages weight settings or domains equally; the overall shopping match rate is instead pooled across expected products.

Table 10 details the evaluation splits, metrics, and benchmark-specific configurations.

Table 10: Benchmark protocol.

Benchmark	Split	Metrics	Protocol
BFCL Multi-Turn	Base, missing function, missing parameter, and long context, 200 tasks each	Multi-turn accuracy per setting and Avg	Multi-turn categories of BFCL v3
τ^2-Bench	Airline, retail, and telecom, 50, 114, and 114 tasks	Task success per domain and Avg	Claude Sonnet 4.6 as the simulated user
MCPMark	Filesystem tasks, 40 tasks	pass@1	Services reset before every episode
DeepPlanning-Travel	240 cases, 120 Chinese and 120 English	Commonsense score, personalized score, and composite, each averaged over the two languages	Itineraries parsed by the official parser; one run per task
DeepPlanning-Shopping	120 cases at three difficulty levels	Match rate per level and pooled over all expected products	-
WideSearch	100 English tasks	Row-level and item-level F1	-

E ORCHESTRATION TRAJECTORIES ON BENCHMARKS

E.1 BEHAVIOR STATISTICS

To measure what the two components of MAW change in the behavior of the policy, we read the event logs of the DeepPlanning-Travel runs behind Table 4, whose three settings share the evaluation architecture, the initialization, and the frozen workers, and report the statistics of Table 11.

Table 11: Orchestration behavior on DeepPlanning-Travel for the three settings of Table 4, where O denotes the outcome reward and P the schedule-aware process reward. Workers is the number of registered workers, Assign. the number of assignments, Rounds the number of delegation calls, Conc. the share of assignments launched inside a batch call that starts at least two different workers (%), Same the share of episodes with duplicate task assignments to the same worker within a batch (%),, and Single the share of episodes with exactly one assignment (%). Comp is the composite score over all 240 tasks (%).

Data	Reward	Workers	Assign.	Rounds ↓	Conc. ↑	Same ↓	Single ↓	Comp ↑
Seed pool	O	1.49	2.49	2.11	24.3	1.0	46.2	12.47
MAW-Data	O	3.40	7.30	4.38	56.4	28.8	0.0	12.14
MAW-Data	$O{+}P$	3.44	5.81	1.88	95.2	0.0	1.9	19.84

Seed pool, O. As shown in Table 11, training on the seed pool leaves the orchestrator close to a single agent, since it registers 1.5 workers per episode, issues exactly one assignment in 46% of the episodes, and launches 24% of its assignments concurrently.

MAW-Data, O. Composed data alone changes the decomposition, but does not yield efficient orchestration. The outcome-only orchestrator registers 3.4 role workers, matching MAW-8B's team size, and issues 7.3 assignments per episode. However, it spreads these assignments over 4.4 sequential delegation rounds, launches only 56% concurrently, and redundantly assigns the same task to the same worker within a batch in 29% of episodes, without improving the score.

MAW-Data, $O{+}P$ (ours). The process reward turns this decomposition into an efficient schedule. With the same team size, MAW-8B issues fewer assignments, launches 95% concurrently across 1.9 rounds, and eliminates redundant dispatches of the same task to the same worker within a batch. In 68% of episodes, it completes delegation in a single batch round before submitting, while the composite score rises from 12.14 to 19.84.

E.2 TRAJECTORY COMPARISON ON ONE TASK

To make Table 11 concrete, we compare the three orchestrators through one DeepPlanning-Travel task(id_15). The request asks for four records that the official checker looks up, marked ❶–❹ below, and for one integration step that arranges them into an itinerary with a budget. The four records do not depend on each other, so an ideal team needs one round of four concurrent lookups followed by the submission.

Request DeepPlanning-Travel, English task id_15

I'm planning a two-day trip from Hangzhou to Shaoxing on November 12, 2025, returning on the 13th. Could you please help me plan my itinerary, including transportation, accommodation, meals, and sightseeing? On the day of departure, I'd like to arrive in Shaoxing as early as possible. Could you help me choose the earliest direct train available ❶? That way, I can have more time to enjoy the city upon arrival. For accommodation, please find me the highest-rated hotel in Shaoxing ❷—just one room for two people. We'd like to stay somewhere comfortable. By the way, I'm especially interested in experiencing Shaoxing's natural scenery this time. Please include all recommended nature-based attractions ❸ in the itinerary—don't miss any. Also, there seem to be many restaurants around 'Shaoxing East Lake Scenic Area', and I've heard the views there are amazing. After visiting East Lake, we'd like to have a meal nearby. Could you please arrange a restaurant that has a waiting area ❹? It would be great if we could rest there while waiting for our table. That covers most of my requirements—I believe I've provided all

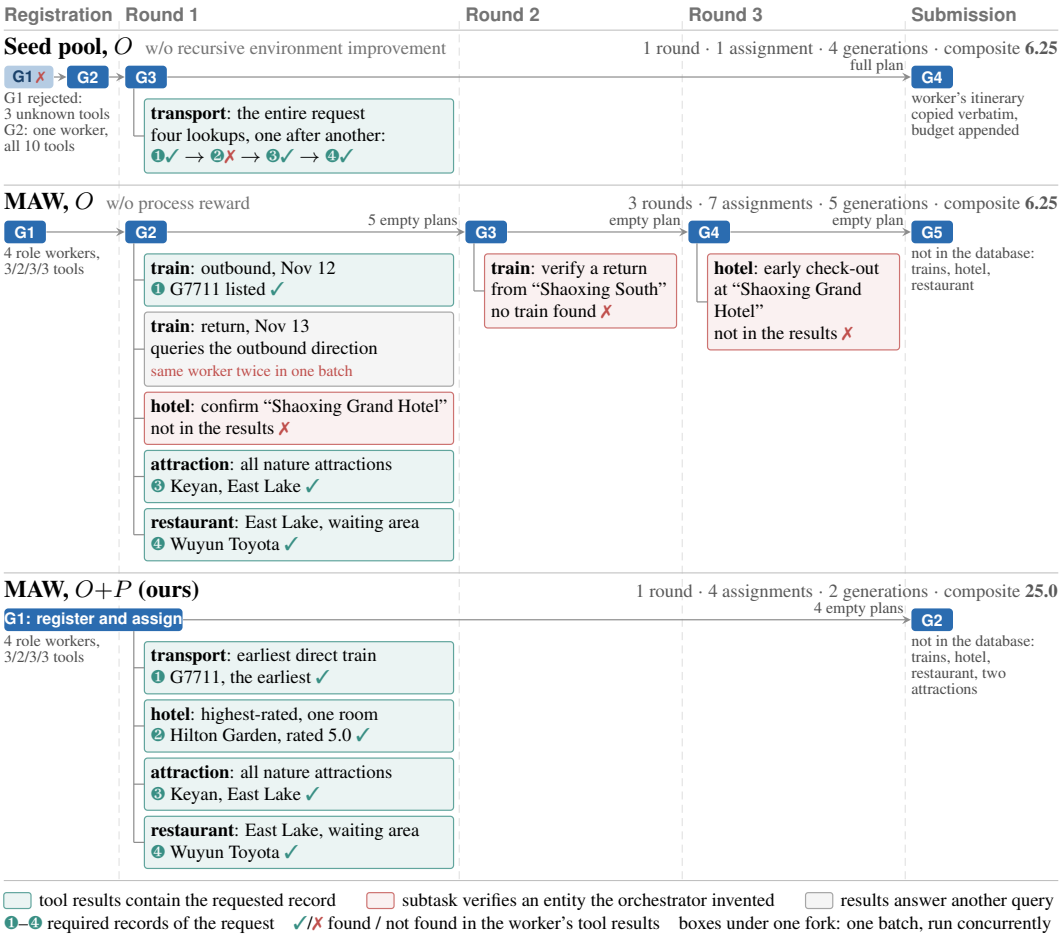


Figure 5: Orchestration trajectories on task id_15 for the three settings in Table 11, read left to right. Blue pills denote orchestrator generations, boxes denote worker assignments, and arrow labels summarize worker returns. Assignments under a shared fork run concurrently. Initial empty-argument probes are omitted.

necessary information. Please go ahead and prepare a detailed itinerary along with a budget plan for me! November 12, 2025 is Wednesday

Records the official checker looks up

❶ train G7711, departing at 06:30 ❷ Shaoxing Yuecheng Hilton Garden Hotel ❸ every attraction of type natural scenery, including Keyan Scenic Area ❹ Wuyun Toyota Staff Restaurant

Figure 5 draws the three episodes step by step.

Seed pool, O. The orchestrator does not decompose the request. Its first registration asks for three tools that do not exist and is rejected, and its second registers one worker with all ten tools. Its only assignment is the request itself, so the worker performs the four lookups one after another and the team adds a hop without adding concurrency. The worker finds ❶, ❸, and ❹, but it misses ❷ because it searches only five-star hotels, while the highest-rated hotel is a four-star hotel rated 5.0. The orchestrator submits the worker's itinerary verbatim and appends a budget.

MAW-Data, O. The orchestrator decomposes the request but does not schedule it. It registers four role workers, and its first batch gives ❶, ❸, and ❹ their own subtasks. The same batch launches the train worker twice, and the second copy, meant for the return train, queries the outbound direction again. The hotel subtask asks to confirm "Shaoxing Grand Hotel", a name that is not in the database

and that first appears in the orchestrator's reasoning, before any worker has returned, as an example of how to copy names exactly, so no subtask asks for 2. Two further rounds verify a return train from "Shaoxing South", a station that no train result lists, and an early check-out at the invented hotel, and both find nothing.

MAW-Data, $O+P$ (ours). The orchestrator gives each record its own subtask. In its first generation it registers the four role workers and dispatches one subtask per record, the four workers find all four records concurrently, and the second generation submits the plan. This is the behavior that the schedule term f_{sch} of Eq. 5 rewards on the composed training tasks and that no benchmark task ever supervised.