E^3-ORCH: TOWARDS EFFECTIVE, EFFICIENT, AND EXTENSIBLE AGENTIC ORCHESTRATION WITH REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic orchestration enables multiple autonomous agents to collaborate on complex tasks through adaptive task decomposition, delegation and execution. However, existing orchestrators largely rely on hand-crafted orchestration logic and prompting strategies, limiting their ability to adapt and generalise across tasks and executor configurations. In this work, we propose E^3-ORCH, a reinforcement learning framework for learning effective, efficient, and extensible agentic orchestration through a *milestone-plan-act* workflow. Rather than planning all subtasks upfront, E^3-ORCH organises execution around *milestones*, a mid-level abstraction that scopes planning and execution around meaningful intermediate objectives, enabling the orchestrator to adapt its decisions as the task evolves. For each milestone, the orchestrator constructs a dependency-aware plan, delegates subtasks to suitable executors, and parallelises independent subtasks for efficient execution. We learn the orchestration policy from execution feedback, using milestone and plan decisions as natural units for fine-grained credit assignment. To estimate their downstream contributions, we construct tree-structured rollouts that compare alternative decisions under shared execution histories. The policy is jointly optimised for task performance and efficiency through complementary rewards for performance, execution cost, and planning completeness, including an uncertainty-aware performance reward that accounts for stochastic downstream outcomes. Across seven benchmarks spanning diverse domains, E^3-ORCH achieves the best task performance under different agent configurations, improving on the strongest baselines by 0.7-3.8 points, and achieves 1.16-$1.59\times$ higher intelligence efficiency. The learned policy also transfers to new agent configurations introduced only at evaluation time, supporting the extensibility of agentic orchestration.

1 INTRODUCTION

Recent advances in large language model (LLM) agents have enabled multi-agent systems (MAS) that combine complementary capabilities to tackle complex tasks through reasoning, tool use, and environment interaction (Yao et al., 2023; Shinn et al., 2023; Qin et al., 2024; Erdogan et al., 2025). Many early MAS rely on task allocations and execution workflows specified before execution, limiting their ability to adapt as tasks evolve during execution (Wu et al., 2023; Hong et al., 2024; Qian et al., 2024; Du et al., 2024; Ke et al., 2025; Dang et al., 2025; Yue et al., 2026). Recent work has therefore shifted toward *agentic orchestration*, moving from fixed workflows to adaptive, execution-aware coordination of agents and tools. This paradigm has been adopted in advanced agentic systems, including Claude Code agent teams (Anthropic, 2026), OpenAI's GPT-5.6 Sol Ultra (OpenAI, 2026a;b), and Kimi K2.5 Agent Swarm (Kimi-Team, 2026). However, existing orchestration policies still largely rely on hand-crafted logic and prompting strategies (Liu et al., 2024; Zhang et al., 2025; Su et al., 2026; Zhang et al., 2026). Their effectiveness therefore remains dependent on human-designed rules and the orchestration capabilities of general-purpose LLMs, potentially limiting adaptation to evolving task states and generalization across tasks and agent configurations.

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.

As illustrated in Figure 1, an agentic orchestrator must dynamically decompose tasks, delegate subtasks, manage execution dependencies, and integrate intermediate outcomes to guide execution (Yue et al., 2026; An et al., 2026). To move beyond hand-crafted strategies, recent work has used reinforcement learning to learn orchestration policies from execution feedback to optimize for objectives such as agent selection, workflow construction, and execution cost (Dang et al., 2025; Yue et al., 2025; Ke et al., 2026). However, many existing approaches often operate at the level of atomic subtasks (Ke et al., 2025; Dang et al., 2025; Yue et al., 2025), either planning a complete workflow upfront (Ke et al., 2026; Su et al., 2026) or executing subtasks sequentially as the task progresses (Li et al., 2025). This fine-grained formulation poses challenges for both orchestration and learning. Planning a complete workflow upfront requires long-horizon reasoning and limits adaptation to intermediate outcomes, while sequential execution can obscure dependency structure and forgo opportunities for parallel execution (Kim et al., 2024; Yue et al., 2026). At the same time, it further complicates credit assignment, as long trajectories of fine-grained decisions make it difficult to attribute final outcomes to individual decisions (Li et al., 2025), particularly under stochastic downstream execution. Moreover, orchestration must balance multiple objectives, including task performance, execution efficiency, and planning completeness. These challenges motivate a higher-level orchestration abstraction that organizes planning and execution into manageable units, supports adaptive and dependency-aware execution, and provides meaningful units for credit assignment.

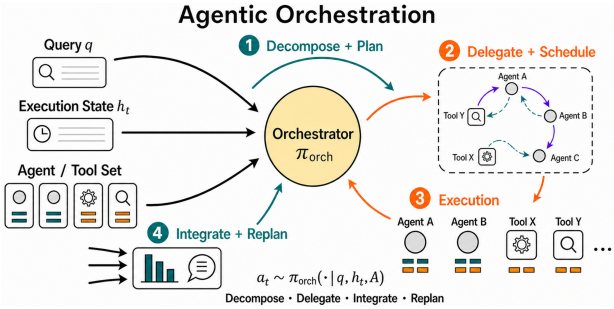

Figure 1: Agentic orchestration adaptively decomposes a task, delegates work, executes agents and tools, and replans from intermediate results.

Effective orchestration requires the right abstraction for both execution and learning. We therefore organize execution around *milestones*: meaningful intermediate objectives that structure a complex task into manageable stages, avoiding the need to plan a long sequence of atomic subtasks at once. Building on this abstraction, we propose E^3-ORCH (Effective, Efficient, and Extensible Agentic Orchestration), a reinforcement learning framework based on an iterative *milestone-plan-act* workflow. For each milestone, the orchestrator constructs a dependency-aware plan that decomposes the milestone into subtasks and their dependencies (Kim et al., 2024; Dang et al., 2025), assigns them to suitable executors, and executes independent subtasks in parallel. The resulting outcomes condition the next milestone and plan, enabling the orchestration structure to be revised as execution proceeds. In this way, milestones scope planning and execution locally while providing natural points for adaptation, avoiding the need to commit to a complete workflow upfront.

Importantly, milestone and plan decisions also provide meaningful units for credit assignment. To estimate the contribution of each decision, we construct *tree-structured rollouts* (Hou et al., 2025) that branch from shared execution histories, allowing alternative milestone and plan decisions to be compared through their downstream outcomes and execution costs. Based on these comparisons, we define three complementary decision-level rewards: a *performance reward* that accounts for uncertainty in downstream outcome, a *cost reward* that compares execution costs across alternative decisions, and a *planning completeness reward* that assesses whether plans are well-formed and executable. We normalize these rewards to jointly optimize the orchestration policy for performance effectiveness, intelligence efficiency, and planning completeness.

We evaluate E^3-ORCH on seven benchmarks spanning mathematical reasoning, embodied decision-making, and deep research under two executor configurations. Across both configurations, E^3-ORCH achieves the best task performance on each benchmark and yields 1.16–1.59$\times$ higher intelligence efficiency than the strongest baseline. Further analyses provide evidence supporting the extensibility of profile-based orchestration. Finally, ablations support the contributions of key components such as tree-structured rollouts to task performance and intelligence efficiency.

In summary, our contributions are threefold: **(1) Orchestration abstraction.** We introduce a *milestone-plan-act* workflow for adaptive, dependency-aware execution and meaningful credit assignment; **(2) Orchestration learning.** We develop tree-structured rollouts and multi-objective

rewards for decision-level policy optimization under uncertain downstream outcomes; and **(3) Empirical performance.** We demonstrate consistent gains of E^3-ORCH in effectiveness, efficiency, and extensibility across diverse tasks and executor configurations.

2 RELATED WORK

Language Agents and Multi-Agent Collaboration. Language agents augment LLM reasoning with planning, tool use, environmental feedback, and memory, which lets models solve interactive and long-horizon tasks (Yao et al., 2023; Shinn et al., 2023; Schick et al., 2023; Qin et al., 2024; Erdogan et al., 2025). Multi-agent systems distribute reasoning and execution across complementary workers through conversation, role specialization, and collaborative aggregation (Wu et al., 2023; Hong et al., 2024; Qian et al., 2024; Du et al., 2024; Qian et al., 2025; Wang et al., 2025). These systems broaden the available expertise and computation, but many of them fix roles, communication protocols, or execution pipelines before execution begins, and such structures are difficult to adapt once intermediate results change the remaining subtasks, their dependencies, or the workers best suited to solve them.

Agentic Orchestration. A second line of work therefore automates how agents and tools are composed, connected, and scheduled, spanning execution-graph and workflow optimization, task-specific agent composition and model routing, and runtime planning and replanning (Zhuge et al., 2024; Zhang et al., 2025; Liu et al., 2024; Yue et al., 2025; Ke et al., 2025; Su et al., 2026; Zhang et al., 2026; Yue et al., 2026). In parallel, supervision and reinforcement learning have been used to improve collaboration and orchestration policies rather than encoding them entirely in prompts or controller logic (Sun et al., 2026; Dang et al., 2025; Ke et al., 2026; Li et al., 2025). Learning orchestration over long horizons nevertheless remains difficult: the execution structure must adapt to evolving task states, outcome-level feedback must be attributed at a meaningful decision granularity, and the three objectives must be optimized without obscuring their distinct signals. We address these requirements by training an LLM orchestrator to construct and revise milestone-level execution graphs, using milestones as the unit of credit assignment and decoupled rewards for multi-objective optimization.

3 METHOD

3.1 PRELIMINARIES

Agentic Orchestration. In this paper, we consider an agentic orchestration system that comprises an *orchestrator* π_{orch} with parameters θ and a pool of executors $\mathcal{A} = \{a_1, \ldots, a_K\}$. with heterogeneous operational profiles Each executor a_k is an LLM agent described by an operational profile $\phi_k = (\mathbf{c}_k, \kappa_k, \mathcal{T}_k)$, summarizing its capabilities, execution cost, and available tools. The resulting profile pool $\Phi = \{\phi_1, \ldots, \phi_K\}$ can incorporate additional executor metadata without changing the orchestration workflow. Given a query q, the orchestrator proposes an intermediate objective as a milestone, decomposes it into subtasks with execution dependencies as a plan, and delegates each subtask to an executor chosen from Φ. A subtask runs once its dependencies are satisfied, so independent subtasks execute in parallel. The orchestrator then integrates their results and decides whether to initiate another milestone-plan-act round or terminate with a final answer. Repeating this loop produces a trajectory τ of orchestration decisions and execution outcomes.

Training Objective. We update only the orchestrator parameters θ, while the executors remain frozen and are treated as part of the execution environment. An orchestrator is judged by the task performance $U(\tau)$ its trajectory achieves, the execution cost $C(\tau)$ it incurs, and the completeness $Q(\tau)$ of the decompositions and plans it produces, that is, whether they are well formed and executable. Our objective is to maximize $U(\tau)$ and $Q(\tau)$ while minimizing $C(\tau)$.

3.2 MILESTONE-PLAN-ACT ORCHESTRATION

Task decomposition is central to orchestration, as it defines the units of work to be planned, scheduled, and executed. However, planning all atomic subtasks in advance is difficult because intermediate outcomes can change their boundaries and dependencies, while long execution chains complicate agent scheduling and result integration. E^3-ORCH therefore adopts an iterative *milestone-plan-act* workflow centered on *milestones*, i.e., intermediate objectives prompted to be concrete and

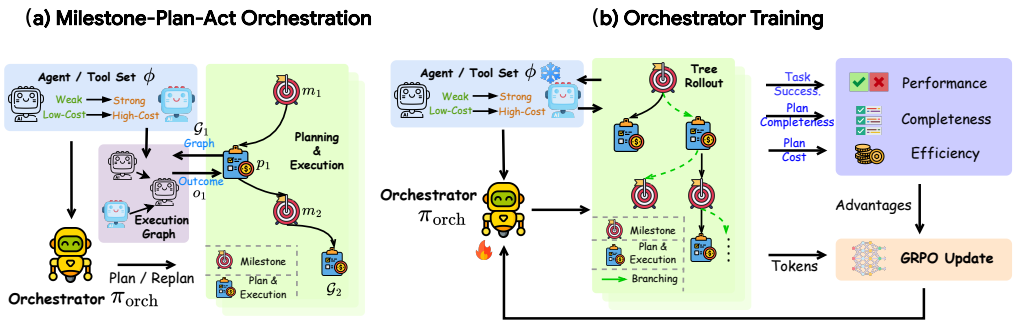

Figure 2: Overview of E^3-ORCH. **(a) Milestone-Plan-Act orchestration.** The orchestrator iteratively proposes milestones, plans dependency-aware execution graphs, delegates to agents and tools, and integrates outcomes for replanning. **(b) Orchestration training.** E^3-ORCH branches over orchestration decisions to form tree-structured rollouts, evaluates alternative orchestration decisions using performance, cost, and completeness rewards.

independently verifiable, which scope task decomposition and parallel scheduling. As illustrated in Figure 2 (a), each iteration i proposes a milestone, plans and executes its subtasks, and aggregates their outputs to guide subsequent planning.

Milestone Proposal. Conditioned on the query q, the current execution history $h_{i-1} = (m_1, p_1, o_1, \ldots, m_{i-1}, p_{i-1}, o_{i-1})$, and the pool profile Φ, the orchestrator proposes the next milestone $m_i \sim \pi_{\text{orch}}(\cdot \mid q, h_{i-1}, \Phi)$ using the prompt in § A.1. Milestones are generated incrementally, allowing subsequent objectives to adapt to prior execution outcomes.

Plan Generation. The orchestrator then plans the selected milestone as $p_i \sim \pi_{\text{orch}}(\cdot \mid q, h_{i-1}, m_i, \Phi)$ (§ A.2), parsed into a directed acyclic graph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$ (Kim et al., 2024; Dang et al., 2025). Each node $v_{ij} \in \mathcal{V}_i$ pairs a subtask with its assigned executor $a(v_{ij}) \in \mathcal{A}$, which may invoke tools from $\mathcal{T}_{a(v_{ij})}$ and return an output o_{ij}. Each edge $(v_{ij}, v_{ik}) \in \mathcal{E}_i$ records that v_{ik} consumes the output of v_{ij}. Each plan thus jointly specifies the subtask decomposition, dependency structure, and executor assignments, enabling parallel execution subject to the milestone's dependency constraints.

Aggregation. The execution engine invokes every node whose dependencies are satisfied, running independent nodes in parallel, and aggregates the node outputs into a milestone-level outcome $o_i = \text{Aggregate}(\{o_{ij} : v_{ij} \in \mathcal{V}_i\})$ using Prompt A.4. The orchestrator appends o_i to the execution history and decides whether to initiate another orchestration round or terminate and answer q. The complete execution trajectory is formulated as $\tau = (m_1, p_1, o_1, \ldots, m_n, p_n, o_n)$, while the milestone proposals m_i and plan generations p_i are the major explicit orchestration actions to be optimized while other orchestration decisions are implicitly learned during training.

3.3 ORCHESTRATION TRAINING

3.3.1 TREE-STRUCTURED ORCHESTRATION ROLLOUTS

A common RL training setup (Shao et al., 2024) independently samples complete trajectories and uses terminal rewards to guide policy updates. For orchestration decisions, differences in their terminal rewards may arise from different preceding histories, subsequent decisions, or stochastic executor behavior, making it difficult to isolate the decision contribution. We therefore adopt tree-structured rollouts (Hou et al., 2025) that branch at selected decisions under a shared execution history and regenerate downstream trajectories for each alternative decision, as illustrated in Figure 3. The resulting branches provide multiple downstream outcome samples for estimating the performance of intermediate decisions, while sibling branches sharing the same execution history provide matched alternatives for relative cost comparison.

Forest Initialization. For each query q we sample B complete trajectories in parallel, using the milestone-plan-act workflow described in § 3.2. These sampled trajectories become the roots of B trees as illustrated on the left of Figure 3.

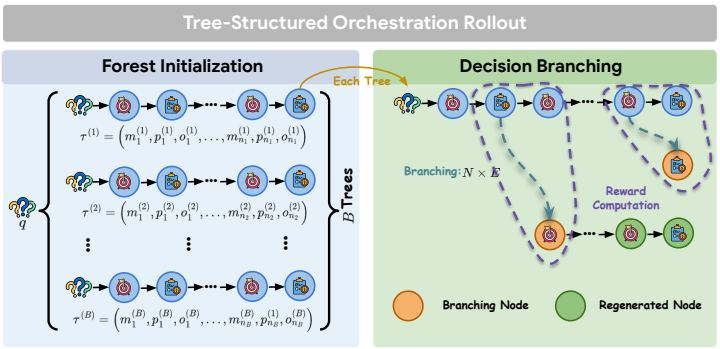

Figure 3: Illustration of the tree-structured orchestration rollout used to train E^3-ORCH. The procedure produces up to $B \times N \times E$ newly expanded trajectories.

Decision Branching. Within each tree we randomly select N orchestration decision nodes including both milestone and plan nodes as branching points. Branching keeps the history preceding the node, resamples the decision there, and regenerates every downstream decision and execution conditioned on the new outcomes. A milestone branch resamples the milestone together with its plan, whereas a plan branch holds the milestone fixed and regenerates only its execution graph. We choose to branch at milestones and plans to explore along the two orchestration decision dimensions of task decomposition and execution strategy, respectively. We repeat this expansion for E rounds and add every valid branch to the forest, which yields up to $B \times N \times E$ new trajectories, as shown on right of Figure 3.

3.3.2 REWARDS DESIGN

Based on the tree-structured rollouts, we design three rewards to evaluate individual orchestration decisions in terms of task performance, intelligence efficiency, and planning completeness. We use ν to denote a decision node representing either a milestone or a plan conditioned on the preceding execution history.

Performance Reward evaluates an orchestration decision's task performance based on the outcomes of its downstream execution. The downstream task performance is stochastic, as subsequent decisions and executor behavior may vary. The empirical success rate alone does not capture the uncertainty arising from limited samples, particularly when decision nodes have different numbers of descendant trajectories. We therefore design a probabilistic performance reward that accounts for uncertainty in each decision's success probability.

Let $\hat{y}(\tau)$ be the final answer of trajectory τ and y^* the reference answer. We define the trajectory-level success indicator as $r_{\text{perf}}(\tau) = \mathbb{I}[\text{Eval}(\hat{y}(\tau); q, y^*) = 1]$, which equals 1 if the final answer is judged correct and 0 otherwise. Here, Eval denotes the benchmark-specific correctness evaluator described in § 4.1. For a decision node ν, let n_ν denote the number of descendant trajectories and k_ν the number of successful ones. We place a uniform prior on its success probability $\psi(\nu)$ and update it using these observed outcomes. To obtain a scalar reward, we compute the probability that this success probability exceeds that of a reference posterior corresponding to zero successes in the same number of attempts:

$$\psi_1 \sim \text{Beta}(k_\nu + 1, n_\nu - k_\nu + 1), \quad \psi_0 \sim \text{Beta}(1, n_\nu + 1),$$
$$R_{\text{perf}}(\nu) = 1 - \frac{\text{B}(k_\nu + 1,\, 2n_\nu - k_\nu + 2)}{\text{B}(k_\nu + 1,\, n_\nu - k_\nu + 1)}, \tag{1}$$

where $\text{B}(\cdot, \cdot)$ denotes the beta function. By comparing the posterior distributions, this reward incorporates uncertainty and quantifies the evidence that the decision outperforms a zero-success baseline ψ_0.

Cost Reward evaluates the downstream execution cost of an orchestration decision, corresponding to the cost component of intelligence efficiency. To control for differences in execution history, we compare costs among sibling decisions sharing the same execution state. For a decision node ν, let $C(\nu)$ denote its downstream execution cost and $\mathcal{S}(\nu)$ denote its sibling set. We define the cost reward by comparing this cost with the sibling average and normalizing the difference by the sibling standard deviation:

$$R_{\text{cost}}(\nu) = \frac{\mu(\{C(\nu') : \nu' \in \mathcal{S}(\nu)\}) - C(\nu)}{\sigma(\{C(\nu') : \nu' \in \mathcal{S}(\nu)\}) + \delta}, \tag{2}$$

where $\mu(\cdot)$ and $\sigma(\cdot)$ denote the average and standard deviation of sibling costs, respectively, and $\delta > 0$ is a small constant for numerical stability. This reward is positive when a decision incurs lower downstream execution cost than the sibling average and negative vice versa.

Completeness Reward evaluates an orchestration decision's structural validity in terms of format validity and executability. Format validity requires milestone and plan outputs to contain all required fields and conform to the expected schema. For plans, structural validity further requires all dependency references to be valid and acyclic, while assignment validity requires every step to name an executor in the available pool. A plan is executable when these constraints are satisfied and its execution successfully produces a milestone-level outcome. We define the completeness reward as a binary indicator of whether both requirements are satisfied:

$$R_{\text{comp}}(\nu) = \mathbb{I}[\nu \text{ is well formed and executable}] \in \{0, 1\}. \tag{3}$$

A decision receives a reward of 1 when both requirements are met and 0 otherwise, providing a validity signal at each decision node to discourage malformed or non-executable outputs during training.

3.3.3 Policy Optimization

The three rewards are computed within its rollout tree. Each reward is then normalized across nodes of the same decision type in the entire forest, and the resulting advantages are combined to guide policy optimization.

Aggregating rewards before advantages normalization can obscure their individual learning signals. We therefore compute advantages separately for each reward and combine them for GRPO optimization (Liu et al., 2026). For $c \in \{\text{perf}, \text{cost}, \text{comp}\}$, let $\mathcal{H}(\nu)$ denote the nodes of the same decision type as ν across all trees generated for query q. The advantage of node ν for reward R_c is computed as:

$$A_c(\nu) = \frac{R_c(\nu) - \mu(\{R_c(\nu') : \nu' \in \mathcal{H}(\nu)\})}{\sigma(\{R_c(\nu') : \nu' \in \mathcal{H}(\nu)\}) + \delta}. \tag{4}$$

The final advantage for node ν is a weighted sum of the normalized advantages as

$$A(\nu) = \lambda_{\text{perf}} A_{\text{perf}}(\nu) + \lambda_{\text{cost}} A_{\text{cost}}(\nu) + \lambda_{\text{comp}} A_{\text{comp}}(\nu), \tag{5}$$

where λ_{perf}, λ_{cost}, and λ_{comp} are corresponding weights for each reward. Finally, the orchestrator is updated with the clipped GRPO objective (Shao et al., 2024). At decision node ν, the state s_ν consists of the query, execution history, and executor profiles Φ, while the action z_ν is the generated token sequence specifying a milestone or a plan.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate E^3-Orch on seven benchmarks spanning three task families: *mathematical reasoning* (AIME24, AIME25, MATH500 (Hendrycks et al., 2021)), *embodied decision-making* (ALFWorld (Shridhar et al., 2021), WebShop (Yao et al., 2022)), and *deep research* (GAIA (Mialon et al., 2024), Humanity's Last Exam (HLE) (Phan et al., 2025)). For training, we collect approximately 7,000 instances from DAPO-Math (Yu et al., 2025), OpenScienceReasoning-2 (NVIDIA, 2025), ALFWorld, WebShop, Natural Questions (Kwiatkowski et al., 2019), HotpotQA (Yang et al., 2018), and WebDancer (Wu et al., 2025) (§ A).

Baselines. We compare against three groups of systems: *single-agent methods* (Direct Reasoning, ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2023)); *multi-agent collaboration methods* (AutoGen (Wu et al., 2023), Multi-Agent Debate (Du et al., 2024)); and *agentic orchestration methods* (MAS-ZERO (Ke et al., 2025), DAAO (Su et al., 2026), AgentFlow (Li et al., 2025), Puppeteer (Dang et al., 2025), MAS-Orchestra (Ke et al., 2026), MasRouter (Yue et al., 2025)). All methods share the same executor backbones and tool interfaces, and every baseline that includes an orchestrator uses the same orchestrator backbone and training settings as E^3-Orch. Specifically, the rollout group size is aligned with that in the tree-structured rollout used by E^3-Orch. See §§ A.10 and A.11 for more details and differences with E^3-Orch.

Implementation Details. We instantiate the orchestrator π_{orch} with Qwen2.5-7B-Instruct (Yang et al., 2024) and train it with GRPO on A100 GPUs using the rLLM framework (Tan et al., 2025). We

Table 1: Task performance of E^3-ORCH and the baselines. The **Training** column indicates whether the orchestrator is learned. Results are averaged over 32 runs for AIME24/25 and 3 runs for the remaining benchmarks, where subscripts are standard deviations over those runs. Bold and underlined entries denote the best and second-best results within each executor block, respectively.

Method	Training	Mathematical		Embodied		Deep Research	
		AIME24/25	MATH500	ALFWorld	WebShop	GAIA	HLE
❄ *Executor: Qwen2.5-7B-Instruct*							
DIRECT REASONING	✘	$10.7_{\pm 0.3}/9.3_{\pm 0.4}$	$72.1_{\pm 0.5}$	$14.8_{\pm 0.4}$	$8.1_{\pm 0.3}$	$3.5_{\pm 0.2}$	$1.1_{\pm 0.3}$
REACT	✘	$16.7_{\pm 0.4}/13.3_{\pm 0.5}$	$77.0_{\pm 0.4}$	$31.2_{\pm 0.6}$	$19.5_{\pm 0.4}$	$3.1_{\pm 0.3}$	$1.5_{\pm 0.2}$
REFLEXION	✘	$22.3_{\pm 0.5}/18.7_{\pm 0.3}$	$81.7_{\pm 0.6}$	$42.7_{\pm 0.5}$	$28.8_{\pm 0.5}$	$4.5_{\pm 0.4}$	$1.8_{\pm 0.3}$
AUTOGEN	✘	$16.7_{\pm 0.3}/14.8_{\pm 0.4}$	$79.5_{\pm 0.5}$	$52.4_{\pm 0.4}$	$41.9_{\pm 0.6}$	$8.3_{\pm 0.5}$	$2.7_{\pm 0.4}$
MULTI-AGENT DEBATE	✘	$18.5_{\pm 0.4}/15.2_{\pm 0.3}$	$80.1_{\pm 0.4}$	$55.3_{\pm 0.7}$	$44.7_{\pm 0.5}$	$9.1_{\pm 0.4}$	$2.9_{\pm 0.3}$
MAS-ZERO	✘	$29.5_{\pm 0.6}/25.0_{\pm 0.5}$	$85.1_{\pm 0.5}$	$58.2_{\pm 0.6}$	$49.7_{\pm 0.4}$	$11.5_{\pm 0.5}$	$3.0_{\pm 0.4}$
AGENTFLOW	✔	$32.8_{\pm 0.5}/28.0_{\pm 0.4}$	$88.1_{\pm 0.4}$	$65.6_{\pm 0.5}$	$57.3_{\pm 0.6}$	$16.2_{\pm 0.4}$	$6.6_{\pm 0.3}$
PUPPETEER	✔	$30.5_{\pm 0.4}/26.0_{\pm 0.5}$	$85.9_{\pm 0.6}$	$61.2_{\pm 0.4}$	$54.1_{\pm 0.5}$	$14.3_{\pm 0.6}$	$4.1_{\pm 0.5}$
MAS-ORCHESTRA	✔	$33.5_{\pm 0.4}/27.0_{\pm 0.6}$	$86.7_{\pm 0.5}$	$68.8_{\pm 0.5}$	$58.9_{\pm 0.4}$	$17.2_{\pm 0.5}$	$3.5_{\pm 0.4}$
MASROUTER	✔	$28.2_{\pm 0.5}/25.3_{\pm 0.4}$	$84.1_{\pm 0.4}$	$63.1_{\pm 0.5}$	$58.2_{\pm 0.5}$	$15.1_{\pm 0.4}$	$5.8_{\pm 0.5}$
DAAO	✔	$30.2_{\pm 0.6}/26.0_{\pm 0.4}$	$85.9_{\pm 0.5}$	$60.1_{\pm 0.6}$	$51.3_{\pm 0.4}$	$12.1_{\pm 0.5}$	$5.4_{\pm 0.4}$
E^3-ORCH	✔	$\mathbf{35.0}_{\pm 0.4}/\mathbf{31.5}_{\pm 0.5}$	$\mathbf{90.4}_{\pm 0.4}$	$\mathbf{70.2}_{\pm 0.5}$	$\mathbf{62.7}_{\pm 0.6}$	$\mathbf{20.5}_{\pm 0.4}$	$\mathbf{7.8}_{\pm 0.3}$
❄ *Executor: GPT-5-mini*							
DIRECT REASONING	✘	$48.3_{\pm 0.5}/40.1_{\pm 0.4}$	$89.7_{\pm 0.3}$	$84.3_{\pm 0.5}$	$73.5_{\pm 0.6}$	$26.5_{\pm 0.4}$	$14.7_{\pm 0.5}$
REACT	✘	$54.2_{\pm 0.6}/45.1_{\pm 0.5}$	$92.6_{\pm 0.4}$	$87.1_{\pm 0.4}$	$77.2_{\pm 0.5}$	$29.1_{\pm 0.5}$	$16.8_{\pm 0.4}$
REFLEXION	✘	$60.7_{\pm 0.4}/50.2_{\pm 0.6}$	$94.8_{\pm 0.5}$	$88.3_{\pm 0.6}$	$79.1_{\pm 0.4}$	$30.6_{\pm 0.3}$	$17.9_{\pm 0.5}$
AUTOGEN	✘	$56.0_{\pm 0.5}/47.3_{\pm 0.4}$	$93.4_{\pm 0.4}$	$89.5_{\pm 0.5}$	$80.8_{\pm 0.5}$	$31.7_{\pm 0.6}$	$18.4_{\pm 0.4}$
MULTI-AGENT DEBATE	✘	$58.1_{\pm 0.4}/49.0_{\pm 0.5}$	$94.1_{\pm 0.5}$	$90.2_{\pm 0.4}$	$81.9_{\pm 0.6}$	$32.5_{\pm 0.4}$	$18.9_{\pm 0.5}$
MAS-ZERO	✘	$61.5_{\pm 0.5}/52.0_{\pm 0.4}$	$95.1_{\pm 0.4}$	$90.2_{\pm 0.5}$	$82.3_{\pm 0.5}$	$33.5_{\pm 0.5}$	$19.1_{\pm 0.4}$
AGENTFLOW	✔	$69.4_{\pm 0.4}/59.0_{\pm 0.5}$	$97.1_{\pm 0.3}$	$93.2_{\pm 0.4}$	$86.5_{\pm 0.5}$	$42.4_{\pm 0.5}$	$23.6_{\pm 0.4}$
PUPPETEER	✔	$67.2_{\pm 0.5}/56.0_{\pm 0.6}$	$96.3_{\pm 0.4}$	$92.5_{\pm 0.6}$	$85.1_{\pm 0.4}$	$40.1_{\pm 0.4}$	$22.4_{\pm 0.5}$
MAS-ORCHESTRA	✔	$68.5_{\pm 0.6}/61.5_{\pm 0.4}$	$97.3_{\pm 0.3}$	$92.0_{\pm 0.5}$	$87.9_{\pm 0.4}$	$43.1_{\pm 0.5}$	$24.1_{\pm 0.3}$
MASROUTER	✔	$58.2_{\pm 0.4}/48.6_{\pm 0.5}$	$94.1_{\pm 0.5}$	$88.9_{\pm 0.4}$	$79.9_{\pm 0.5}$	$31.2_{\pm 0.5}$	$18.1_{\pm 0.6}$
DAAO	✔	$62.3_{\pm 0.5}/53.1_{\pm 0.4}$	$95.5_{\pm 0.4}$	$91.0_{\pm 0.5}$	$83.1_{\pm 0.4}$	$34.2_{\pm 0.6}$	$19.5_{\pm 0.4}$
E^3-ORCH	✔	$\mathbf{73.2}_{\pm 0.5}/\mathbf{63.8}_{\pm 0.4}$	$\mathbf{98.0}_{\pm 0.2}$	$\mathbf{95.1}_{\pm 0.4}$	$\mathbf{89.4}_{\pm 0.4}$	$\mathbf{45.7}_{\pm 0.5}$	$\mathbf{26.2}_{\pm 0.4}$

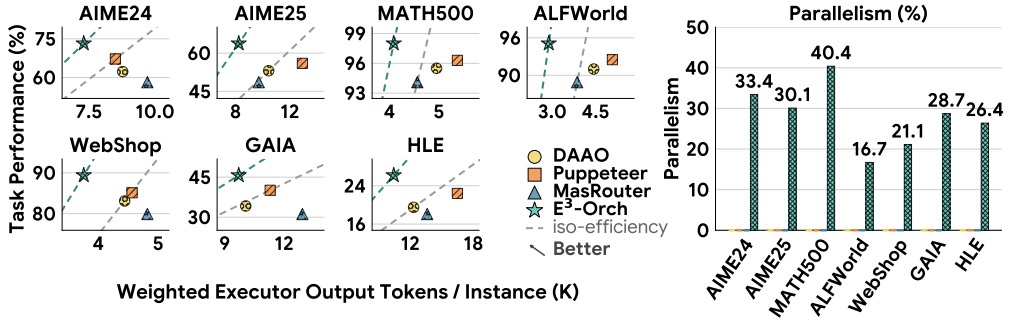

Figure 4: Intelligence efficiency and execution parallelism. **(Left)** E^3-ORCH achieves the Pareto frontier in intelligence efficiency with the highest task performance and lowest token cost. **(Right)** E^3-ORCH executes 16.7–40.4% of its calls in parallel, compared with 0% for the evaluated baselines for superior execution efficiency.

summarize the training hyperparameters in Table 3. The executor pool uses Qwen2.5-7B-Instruct during training, and at evaluation time we additionally consider a GPT-5-mini configuration (OpenAI, 2025). Executors are exposed to the orchestrator through two anonymous profiles that differ in both capability and cost, follow REACT (Yao et al., 2023), and share a common tool library (§ A.8). We verify answers with Math-Verify (Hugging Face, 2024) for mathematical tasks and CompassVerifier (Liu et al., 2025) otherwise (§ A.6).

4.2 MAIN RESULTS

Performance Effectiveness. E^3-ORCH achieves the best result on each benchmark under both executor configurations (Table 1), improving on the strongest competing result in each column by 0.7 to 3.8 points. The gains span mathematical reasoning, embodied decision-making, and deep research, which indicates that the learned orchestration policy is not tied to a single task structure. By

Table 2: Ablation results with Qwen2.5-7B-Instruct executors, averaged within each benchmark family. We report task performance (↑) and token cost (↓), the weighted executor output tokens per instance in thousands, their ratio is the intelligence efficiency of Figure 4.

Method	Mathematical		Embodied		Deep Research	
	Perf.	Tokens (K)	Perf.	Tokens (K)	Perf.	Tokens (K)
E^3-ORCH	**52.3**$_{\pm 0.4}$	**6.2**$_{\pm 0.5}$	**66.5**$_{\pm 0.5}$	**3.3**$_{\pm 0.4}$	**14.2**$_{\pm 0.4}$	**10.4**$_{\pm 0.4}$
w/ Standard Rollouts	48.8$_{\pm 0.5}$	7.8$_{\pm 0.9}$	62.3$_{\pm 0.6}$	4.4$_{\pm 0.6}$	10.0$_{\pm 0.3}$	11.8$_{\pm 1.0}$
w/o Decision Branching	49.2$_{\pm 0.4}$	7.3$_{\pm 0.6}$	65.1$_{\pm 0.5}$	4.8$_{\pm 0.4}$	12.5$_{\pm 0.5}$	11.3$_{\pm 0.5}$
w/ Mean Performance Reward	50.1$_{\pm 0.6}$	6.5$_{\pm 0.3}$	63.2$_{\pm 0.4}$	3.3$_{\pm 0.3}$	12.5$_{\pm 0.4}$	10.9$_{\pm 0.9}$
w/o Cost Reward	50.6$_{\pm 0.5}$	11.3$_{\pm 0.9}$	65.1$_{\pm 0.6}$	5.7$_{\pm 0.6}$	13.5$_{\pm 0.6}$	12.9$_{\pm 0.6}$
w/ Scalarized Reward	50.5$_{\pm 0.4}$	7.2$_{\pm 0.6}$	64.5$_{\pm 0.5}$	3.5$_{\pm 0.4}$	12.9$_{\pm 0.5}$	10.9$_{\pm 0.9}$

replacing Qwen2.5-7B-Instruct executors with GPT-5-mini, we observe that every method's absolute performance improves, while E^3-ORCH remains the best in every column. This indicates the learned orchestration ability can be transferred to a new executor configuration without any change to the milestone-plan-act interface. Scaling behavior across 3B and 7B orchestrator backbones is reported in § B.1.

Intelligence Efficiency. We measure intelligence efficiency as task performance per cost-weighted executor output token, reflecting the goal of completing tasks accurately with fewer tokens (OpenAI, 2026c) (see details in § A.13). As shown on the left of Figure 4, E^3-ORCH achieves the Pareto frontier with the highest task performance and lowest weighted token cost among the methods shown on all seven benchmarks. Its performance per token is 1.16–1.59× higher than that of the best baseline, indicating the superior intelligence efficiency. As shown on the right of Figure 4, E^3-ORCH executes 16.7–40.4% of its executor calls in parallel, compared with 0% for the evaluated baselines. This suggests that the learned orchestrator can identify independently executable subtasks through task decomposition and organize them for concurrent execution. Overlapping independent work can shorten task execution time, consistent with the lower wall-clock times reported in § B.2.

Executor Extensibility. We evaluate the extensibility of E^3-ORCH along two dimensions: *breadth*, which adds a third executor to the pool, and *depth*, which adds a capability attribute to each executor profile (Figure 5). Under both extensions, E^3-ORCH improves task performance relative to its base configuration and achieves ≈ 1.95× intelligence efficiency compared to AgentFlow. By contrast, MasRouter loses 7.8–9.0 performance points relative to its base configuration and achieves only 0.71–0.75× AgentFlow's efficiency. These results suggest that E^3-ORCH can benefit from both a broader executor pool and more detailed capability profiles. Detailed extensibility results can be found in § B.3.

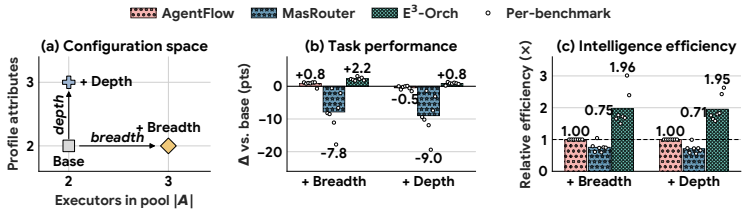

Figure 5: Extension across executor configurations. (**Left**) *Breadth* adds a third executor (Table 6) and *depth* adds a capability attribute to each profile (Table 7). (**Middle**) Task performance change relative to the base configurations.(**Right**) intelligence efficiency relative to AgentFlow (dashed line). Bars show means across benchmarks, and dots show individual benchmark results.

4.3 ABLATION STUDY

Table 2 isolates the contributions of the rollout structure, the individual rewards, and the decoupled normalization, reporting task performance and token cost separately so that each component's effect on intelligence efficiency is visible. Additional analyses of the orchestration abstraction and completeness reward are provided in §§ B.4 and B.5. (1) *Replacing tree-structured rollouts with independent standard rollouts* is the most damaging ablation on both axes, retaining only 0.62–0.74× the intelligence efficiency of the full method, because without shared prefixes neither reward can be computed against a matched alternative. (2) *Removing decision branching* is milder (0.80–0.85×), confirming that branching at milestone and plan decisions supplies useful variation beyond that of

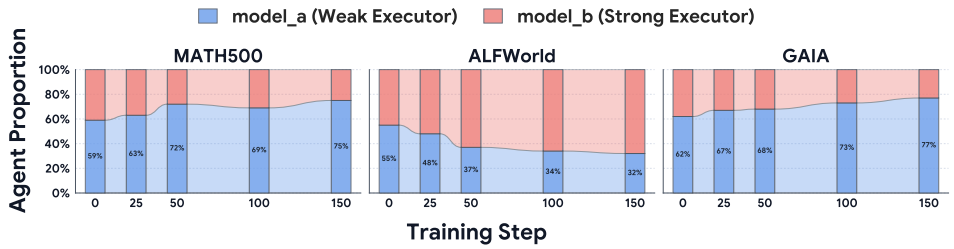

Figure 6: Executor usage during training. `model_a` is a weaker, lower-cost executor, whereas `model_b` is a stronger and more expensive executor.

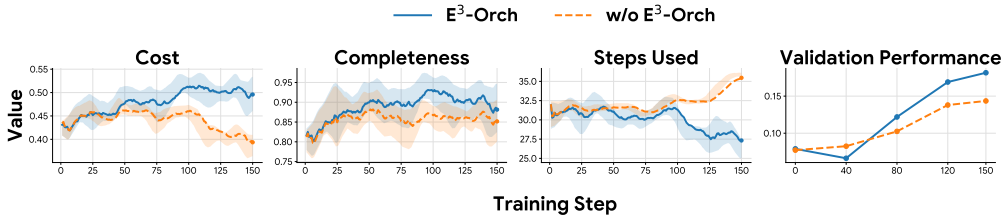

Figure 7: Training dynamics of E^3-ORCH and the standard-GRPO baseline. *Cost* is the cost reward, *Completeness* is the fraction of orchestration decisions that are well formed and executable, and *Steps Used* is the number of execution steps.

independent initial trajectories alone. The two outcome-based rewards then affect complementary axes. (3) *Replacing the uncertainty-aware performance reward in Equation* (1) *with the mean performance of descendant trajectories* costs 1.7–3.3 performance points while leaving token cost almost untouched, consistent with its motivation that nodes supported by different numbers of descendants should not receive identical confidence. (4) *Removing* R_{cost} instead causes the smallest performance drop of any ablation yet inflates token cost by 24–82%, lowering intelligence efficiency to 0.53–0.77×: task performance alone would rank this variant among the best ablations, whereas intelligence efficiency ranks it last. (5) Finally, *combining the three raw rewards before normalizing once* keeps most of the cost benefit but pays 1.3–2.0 performance points for it, falling below *w/o Cost Reward* on task performance in every family, a pattern consistent with reward-signal collapse (Liu et al., 2026), where aggregation before normalization makes distinct reward combinations harder for the policy to distinguish.

4.4 Behavior Analysis of the Learned Orchestrator

Figure 6 shows that the optimized policy develops benchmark-dependent executor preferences. Over training, the share of calls to the weaker and cheaper `model_a` rises from 59% to 75% on MATH500 and from 62% to 77% on GAIA, but falls from 55% to 32% on ALFWorld in favor of the stronger `model_b`, so the policy defaults to neither the cheapest nor the strongest executor and instead allocates according to task difficulty and the capability–cost profiles. The corresponding training dynamics in Figure 7 show E^3-ORCH maintaining higher efficiency and completeness than the standard-GRPO baseline while progressively reducing execution steps, ending at roughly 27 steps against 36 for the baseline and, after an initial dip, at higher validation performance. These curves are consistent with the ablations, as tree-structured comparison and separately normalized rewards jointly favor trajectories that are better formed and less computationally redundant. Orchestration behavior cases can be found in § B.6.

5 Conclusion

We introduced E^3-ORCH, which trains an LLM orchestrator with reinforcement learning over a milestone-plan-act workflow to adapt to evolving task states. Tree-structured rollouts compare decisions under shared execution histories, with separately normalized rewards for task performance, execution cost, and planning completeness. Across seven benchmarks, E^3-ORCH achieves the best task performance and higher intelligence efficiency, while benefiting from extension to larger executor pools and richer capability executors. Ablations and behavior analyses support the contributions of each component.

AI USE STATEMENT

We use generative AI tools to refine the writing of this paper for readability and use AI coding assistance for vibing code development. No other disclosure categories apply. We have reviewed all AI-assisted text and take full responsibility for the final content of this work, including all claims and artifacts.

ETHICS STATEMENT

We focus on general scientific research questions and pose no risks to safety, personal security, or privacy. No new datasets are released as part of this study. Additionally, the research does not encompass potentially harmful insights, methods, or applications, nor does it raise issues related to privacy, security, legal compliance, or research integrity. We foresee no ethical risks or conflicts of interest. We are committed to adhering to ethical guidelines throughout the research process.

REPRODUCIBILITY STATEMENT

The orchestration workflow and its decision points are specified in § 3.2, and the training procedure, including the construction of tree-structured rollouts, the three decision-level rewards, and the decoupled normalization, is specified in § 3.3. All prompts used by the orchestrator and the executors are reproduced verbatim in §§ A.1, A.2 and A.4, and the grading prompt is given in § A.6.

The experimental configuration is likewise fully specified. § A.7 gives the exact executor-pool definition exposed to the orchestrator, § A.8 gives the complete interface of every tool together with the sandbox and search backends, § A documents the training and evaluation data sources, and § A.10 documents each baseline and the settings shared across methods, namely the same executor backbones, the same tool interfaces, and for every trained baseline the same orchestrator backbone and rollout group size. Table 3 lists all optimization and decoding hyperparameters, and the orchestrator is trained with the open-source framework and hardware reported in § 4.1.

REFERENCES

B. An, B. Li, B. Wang, B. Zhang, B. L. Wang, C. Feng, C. Wei, C. Xue, C. Zhang, D. Ng, D. Ye, E. Min, F. Chen, F. Liu, F. Yang, F. Ye, G. Sun, H. Ji, H. Xu, H. Yang, H. Ye, H. Zhang, H. Zhao, J. Li, J. Lin, J. Xia, K. Jin, K. Wang, K. Yang, L. Bing, L. Lei, L. Su, Le. Wang, Lu. Wang, N. Wang, Q. Ren, Q. Yang, R. Li, S. Bai, S. Du, S. Li, S. Lin, S. Nie, S. Wang, S. Zhang, S. Z. Wang, T. Ge, Ta. Q. Fang, Ti. Q. Fang, W. Fang, W. Li, W. Zhang, X. Chen, X. Li, X. Tang, X. Wang, X. Xu, X. Zhang, X. Q. Wang, X. Y. Wang, Y. Deng, Y. Gao, Y. Hu, Y. Li, Y. Sui, Y. Wang, Y. Xiao, Y. Zhang, Y. Zhou, Z. Chen, Z. Cheng, Z. Feng, Z. Liang, Z. Liu, and Z. Zhang. Apodex 1.1: Scaling agentic intelligence for complex work. *CoRR*, abs/2608.23283, 2026. 1

Anthropic. Orchestrate teams of Claude Code sessions. https://code.claude.com/docs/en/agent-teams, 2026. Accessed: 2026-09-11. 1

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Ming-Hsuan Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *CoRR*, abs/2502.13923, 2025. A.8

bytedance. Code sandbox: A secure sandbox for running and judging code generated by llms. https://github.com/bytedance/SandboxFusion, 2024. A.8

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration. In *NeurIPS*, 2025. 1, 1, 2, 3.2, 4.1, A.10

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *ICML*, Proceedings of Machine Learning Research, pp. 11733–11763. PMLR / OpenReview.net, 2024. 1, 2, 4.1, A.10

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In *ICML*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. 1, 2

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks*, 2021. 4.1, A.9.2

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for A multi-agent collaborative framework. In *ICLR*. OpenReview.net, 2024. 1, 2

Zhenyu Hou, Ziniu Hu, Yujiang Li, Rui Lu, Jie Tang, and Yuxiao Dong. TreeRL: LLM reinforcement learning with on-policy tree search. In *ACL (1)*, pp. 12355–12369. Association for Computational Linguistics, 2025. 1, 3.3.1

Hugging Face. Math-verify. https://github.com/huggingface/Math-Verify, 2024. 4.1

Zixuan Ke, Austin Xu, Yifei Ming, Xuan-Phi Nguyen, Caiming Xiong, and Shafiq Joty. MAS-ZERO: designing multi-agent systems with zero supervision. *CoRR*, abs/2505.14996, 2025. 1, 1, 2, 4.1, A.10

Zixuan Ke, Yifei Ming, Austin Xu, Ryan Chin, Xuan-Phi Nguyen, Prathyusha Jwalapuram, Jiayu Wang, Semih Yavuz, Caiming Xiong, and Shafiq Joty. Mas-orchestra: Understanding and improving multi-agent reasoning through holistic orchestration and controlled benchmarks. *CoRR*, abs/2601.14652, 2026. 1, 2, 4.1, A.10

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *ICML*, Proceedings of Machine Learning Research, pp. 24370–24391. PMLR / OpenReview.net, 2024. 1, 3.2

Kimi-Team. Kimi K2.5: visual agentic intelligence. *CoRR*, abs/2602.02276, 2026. 1

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Trans. Assoc. Comput. Linguistics*, 7:452–466, 2019. 4.1, A.9.1

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use. *CoRR*, abs/2510.05592, 2025. 1, 2, 4.1, A.10

Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Peter Belcak, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, Yejin Choi, Jan Kautz, and Pavlo Molchanov. GDPO: group reward-decoupled normalization policy optimization for multi-reward RL optimization. *CoRR*, abs/2601.05242, 2026. 3.3.3, 4.3

Shudong Liu, Hongwei Liu, Junnan Liu, Linchen Xiao, Songyang Gao, Chengqi Lyu, Yuzhe Gu, Wenwei Zhang, Derek F. Wong, Songyang Zhang, and Kai Chen. Compassverifier: A unified and robust verifier for llms evaluation and outcome reward. In *EMNLP*, pp. 33466–33494. Association for Computational Linguistics, 2025. 4.1, A.6

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *CoLM*, 2024. 1, 2

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general AI assistants. In *ICLR*. OpenReview.net, 2024. 4.1, A.9.2

NVIDIA. Opensciencereasoning-2. https://huggingface.co/datasets/nvidia/OpenScienceReasoning-2, 2025. 4.1, A.9.1

OpenAI. Gpt-5 is our smartest, fastest, most useful model yet, with built-in thinking that puts expert-level intelligence in everyone's hands. https://openai.com/index/introducing-gpt-5/, 2025. 4.1

OpenAI. Prompt used for "a proof of the cycle double cover conjecture". https://cdn.openai.com/pdf/04d1d1e4-bc75-476a-97cf-49055cd98d31/cdc_prompt.pdf, 2026a. 1

OpenAI. A proof of the cycle double cover conjecture. https://cdn.openai.com/pdf/04d1d1e4-bc75-476a-97cf-49055cd98d31/cdc_proof.pdf, 2026b. 1

OpenAI. How gpt-5.6 fuses frontier intelligence with frontier efficiency. https://openai.com/index/gpt-5-6-frontier-intelligence-efficiency/, 2026c. 4.2

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, Adam Khoja, Ryan Kim, Richard Ren, Jason Hausenloy, Oliver Zhang, Mantas Mazeika, Summer Yue, Alexandr Wang, and Dan Hendrycks. Humanity's last exam. *CoRR*, abs/2501.14249, 2025. 4.1, A.9.2

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development. In *ACL (1)*, pp. 15174–15186. Association for Computational Linguistics, 2024. 1, 2

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 *ICLR*. OpenReview.net, 2025. 2

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *ICLR*. OpenReview.net, 2024. 1, 2

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 *NeurIPS*, 2023. 2

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. 3.3.1, 3.3.3

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*, 2023. 1, 2, 4.1, A.10

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *ICLR*. OpenReview.net, 2021. 4.1, A.8, A.9.1, A.9.2

Jinwei Su, Qizhen Lan, Yinghui Xia, Lifan Sun, Weiyou Tian, Tianyu Shi, and Lewei He. Difficulty-aware agentic orchestration for query-specific multi-agent workflows. In *WWW*, pp. 2060–2070. ACM, 2026. 1, 1, 2, 4.1, A.10

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. Zerosearch: Incentivize the search capability of llms without searching. *CoRR*, abs/2505.04588, 2025. A.8

Lin Sun, Chuang Liu, Can Zhang, Yubin Wu, Weijia Lu, and Ning Wu. AgentPO: Enhancing multi-agent collaboration via reinforcement learning. In *ICLR*. OpenReview.net, 2026. 2

Sijun Tan, Michael Luo, Colin Cai, Tarun Venkat, Kyle Montgomery, Aaron Hao, Tianhao Wu, Arnav Balyan, Manan Roongta, Chenguang Wang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. rllm: A framework for post-training language agents. https://pretty-radio-b75.notion.site/rLLM-A-Framework-for-Post-Training-Language-Agents-21b81902c146819db63cd98a54ba5f31, 2025. Notion Blog. 4.1

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In *ICLR*. OpenReview.net, 2025. 2

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Webdancer: Towards autonomous information seeking agency. *CoRR*, abs/2505.22648, 2025. 4.1, A.9.1

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversation framework. *CoRR*, abs/2308.08155, 2023. 1, 2, 4.1, A.10

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *CoRR*, abs/2412.15115, 2024. 4.1

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*, pp. 2369–2380. Association for Computational Linguistics, 2018. 4.1, A.9.1

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *NeurIPS*, 2022. 4.1, A.8, A.9.1, A.9.2

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *ICLR*. OpenReview.net, 2023. 1, 2, 4.1, 4.1, A.10

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Weinan Dai, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: an open-source LLM reinforcement learning system at scale. *CoRR*, abs/2503.14476, 2025. 4.1, A.9.1

Ling Yue, Kushal Raj Bhandari, Ching-Yun Ko, Dhaval Patel, Shuxin Lin, Nianjun Zhou, Jianxi Gao, Pin-Yu Chen, and Shaowu Pan. From static templates to dynamic runtime graphs: A survey of workflow optimization for LLM agents. *CoRR*, abs/2603.22386, 2026. 1, 1, 2

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In *ACL (1)*, pp. 15549–15572. Association for Computational Linguistics, 2025. 1, 2, 4.1, A.10

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *ICML*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. 1, 2

Xing Zhang, Yanwei Cui, Guanghui Wang, Wei Qiu, Ziyuan Li, Fangwei Han, Yajing Huang, Hengzhi Qiu, Bing Zhu, and Peiyang He. Verified multi-agent orchestration: A plan-execute-verify-replan framework for complex query resolution. *CoRR*, abs/2603.11445, 2026. 1, 2

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *ICML*, Proceedings of Machine Learning Research, pp. 62743–62767. PMLR / OpenReview.net, 2024. 2

Appendix

Contents of Appendix

A Additional Implementation Details **16**

A.1 Prompt for Milestone Generation 16

A.2 Prompt for Plan Generation 16

A.3 Executor Prompt 17

A.4 Prompt for Milestone Outcome Generation 18

A.5 Prompt for Final Answer Generation 18

A.6 Prompt for CompassVerifier 19

A.7 Executor Pool Configuration 20

A.8 Tool Configuration 21

A.9 Dataset Details 27

A.9.1 Training Data 27

A.9.2 Evaluation Benchmarks 27

A.10 Baseline Details 28

A.11 Baselines Implementation Details 29

A.12 Training Parameters 29

A.13 Definition of Intelligence Efficiency 30

B Additional Experimental Results and Analysis **31**

B.1 Scaling the Orchestrator 31

B.2 Wall-Clock Time Analysis 31

B.3 Generalization Across Executor Configurations 32

B.4 Analysis of Orchestration Abstraction 33

B.5 Ablation of Completeness Reward 34

B.6 Case Studies 35

C Limitations and Future Work **38**

A Additional Implementation Details

A.1 Prompt for Milestone Generation

Prompt A.1 gives the prompt used by the orchestrator π_{orch} to generate the next milestone.

Prompt A.1: Prompt for Next Milestone Generation

```
You are a strategic orchestrator driving a task to completion
one milestone at a time.

=== Task ===
{task}

=== Milestones completed so far ===
{milestone_history}

=== Instructions ===
Examine the task and the results of every milestone completed
so far, then decide between the following two options.

Option A. The task is NOT yet fully solved.  Return:
   {"task_solved":  false, "milestone":  "<single imperative
sentence describing the next concrete checkpoint>" }
Guidelines for the next milestone:
- It must be independently verifiable and build directly on
what has already been done.
- Write it as a single, imperative sentence (e.g., "Verify
that the purchased item matches the target specification").
- Do NOT describe sub-steps or implementation details.
- Do NOT add a milestone that merely restates the final answer.
Synthesis is handled automatically.

Option B. The task IS fully solved based on the milestone
results above.  Return:
   {"task_solved":  true, "milestone":  ""}
Choose Option B only when the completed milestones
collectively provide everything needed to answer the original
task with confidence.  Do not stop early.

Respond with a JSON object matching the required schema.
```

A.2 Prompt for Plan Generation

Prompt A.2 gives the prompt used by π_{orch} to construct an execution plan for a selected milestone.

Prompt A.2: Prompt for Plan Generation

```
You are an expert task orchestrator responsible for producing
an executable plan for a single milestone within a larger
task.

=== Context ===
Overall task:
   {task}
Results from previously completed milestones:
   {previous_milestone_results}
Current milestone to accomplish:
   {current_milestone}
```

```
=== Available resources ===
Models (use EXACTLY one of these identifiers per step):
   {available_models}
Model intelligence ranking (weakest -> strongest):
   {model_intelligence_rank}
Model cost ranking (cheapest -> most expensive):
   {model_cost_rank}
Available tools:
   {available_tools}

=== Planning rules ===
1.  Break the milestone into the minimal set of sub-tasks that
together accomplish it.
2.  Each step must have a unique integer "id" starting from
1.
3.  A step's "dependencies" lists the ids of steps that must
complete before it starts.  Use an empty list [] for steps
that can start immediately.
4.  Assign the cheapest model that is sufficiently capable for
each step.  Reserve stronger (more expensive) models only for
reasoning-heavy steps.
5.  Keep sub-task descriptions concise and self-contained.
The executor will receive only the sub-task text plus outputs
from its dependencies.
6.  Prefer parallelism:  steps with no shared dependencies
will run concurrently.
7.  Do NOT add a "summarize results" step.  That is handled
automatically after execution.

=== Output ===
Respond with a JSON object matching the required schema.
Each element of the "steps" list must contain:
- "id":  integer
- "sub_task":  string (what the executor must do)
- "model":  string (exact model identifier from the list
above)
- "dependencies":  list of integer ids
```

A.3 Executor Prompt

Prompt A.3 gives the prompt supplied to an executor $a_k \in \mathcal{A}$ for an assigned plan step.

Prompt A.3: Executor Prompt

```
You are a focused executor agent.  Your job is to accomplish
exactly one sub-task using the tools available to you.

=== Overall task (for context only) ===
{task}

=== Completed milestone results ===
{previous_milestone_results}

=== Outputs from upstream steps in the current plan ===
{previous_execution_logs}

=== Your sub-task ===
{sub_task}

=== Instructions ===
```

```
- Use tools as needed to gather information or perform
actions.
- Think step-by-step, but keep intermediate reasoning brief.
- When you have a confident answer, wrap it in \boxed{<your
answer here>} and stop.  Do not add further commentary after
the boxed answer.
- If you cannot complete the sub-task (e.g.  a required
resource is unavailable), state the reason clearly inside
\boxed{FAILED: <reason>}.
- Do not attempt to accomplish anything beyond the stated
sub-task.
```

A.4 Prompt for Milestone Outcome Generation

Prompt A.4 gives the prompt used to aggregate executor outputs into a milestone-level outcome.

Prompt A.4: Prompt for Milestone Outcome Generation

```
You are a synthesis assistant.  Given the outputs of several
executor steps that together addressed a single milestone,
produce a concise, accurate summary of what was accomplished.

=== Milestone ===
{current_milestone}

=== Step results ===
{plan_results}

=== Instructions ===
- Synthesize the step results into a single coherent answer
for the milestone.
- Include all factual findings that are relevant to the
milestone, omit execution details (tool calls, intermediate
reasoning, errors) unless they materially affect the answer.
- If some steps failed or returned no useful output, note what
is missing and reason from available information.
- State your final milestone answer inside \boxed{<answer>}.
The boxed content should be a standalone, readable summary
(1-5 sentences or a concise structured list if appropriate).
```

A.5 Prompt for Final Answer Generation

Prompt A.5 gives the prompt used to synthesize the final answer from the completed milestone outcomes.

Prompt A.5: Prompt for Final Answer Generation

```
You are a final-answer synthesizer.  You have been given a
task and the summarized results for each milestone that was
executed to accomplish it.  Your role is to produce the
definitive answer to the original task.

=== Original task ===
{task}

=== Milestone results ===
{milestone_results}
```

```
=== Instructions ===
- Integrate all milestone results into a single, complete, and
coherent answer to the original task.
- Present information at the level of detail the task
requires:  a factual lookup warrants a direct answer, an
analytical task warrants structured reasoning.
- Do not expose internal workflow details (milestones, plan
steps, tool usage).
- If any milestone failed or produced insufficient results,
acknowledge the limitation concisely and answer as accurately
as possible from available data.
- Place your final answer inside \boxed{<answer>}.  Everything
outside the box is discarded.
```

A.6 Prompt for CompassVerifier

Prompt A.6 gives the evaluation prompt used by CompassVerifier (Liu et al., 2025).

Prompt A.6: Prompt for CompassVerifier

```
Please as a grading expert, judge whether the final answers
given by the candidates below are consistent with the
standard answers, that is, whether the candidates answered
correctly.

Here are some evaluation criteria:
1.  Please refer to the given standard answer.  You don't need
to re-generate the answer to the question because the
standard answer has been given.  You only need to judge
whether the candidate's answer is consistent with the standard
answer according to the form of the question.  THE STANDARD
ANSWER IS ALWAYS CORRECT AND THE QUESTION IS PERFECTLY VALID.
NEVER QUESTION THEM.
2.  ONLY compare the FINAL ANSWER - COMPLETELY IGNORE any
potential errors in the REASONING PROCESSES.
3.  Some answers may be expressed in different ways, such as
some answers may be a mathematical expression, some answers
may be a textual description, as long as the meaning
expressed is the same.  Before making a judgment, please
understand the question and the standard answer first, and
then judge whether the candidate's answer is correct.
4.  Some answers may consist of multiple items, such as
multiple-choice questions, multiple-select questions,
fill-in-the-blank questions, etc.  Regardless of the question
type, the final answer will be considered correct as long as
it matches the standard answer, regardless of whether the
reasoning process is correct.  For multiple-select questions
and multi-blank fill-in-the-blank questions, all
corresponding options or blanks must be answered correctly
and match the standard answer exactly to be deemed correct.
5.  If the prediction is given with \boxed{}, please ignore
the \boxed{} and only judge whether the candidate's answer is
consistent with the standard answer.
6.  If the candidate's answer is invalid (e.g., incomplete
(cut off mid-response), lots of unnormal repetitive content,
or irrelevant to the question, saying it can't answer the
question because some irresistible factors, like ethical
```

```
issues, no enough information, etc.), select option C
(INVALID).

Please judge whether the following answers are consistent
with the standard answer based on the above criteria.  Grade
the predicted answer of this new question as one of:
A: CORRECT
B: INCORRECT
C: INVALID

Just return the letters "A", "B", or "C", with no text around
it.

Here is your task.  Simply reply with either CORRECT,
INCORRECT, or INVALID. Don't apologize or correct yourself if
there was a mistake, we are just trying to grade the answer.

<Original Question Begin>:
{question}
<Original Question End>

<Standard Answer Begin>:
{gold_answer}
<Standard Answer End>

<Candidate's Answer Begin>:
{llm_response}
<Candidate's Answer End>

Judging the correctness of the candidate's answer:
```

A.7 Executor Pool Configuration

The following YAML example illustrates how we instantiate the executor pool $\mathcal{A}$ and expose its operational profiles to the orchestrator. The orchestrator observes anonymous identifiers, such as `model_a`, together with capability and cost rankings rather than recognizable model names. The same interface can represent additional executors or profile attributes without changing the milestone-plan-act workflow.

```yaml
model_list:
  - model_a
  - model_b

model_names:
  model_a: Qwen/Qwen2.5-7B-Instruct
  model_b: Qwen/Qwen2.5-7B-Instruct

model_urls:
  model_a: xxxx
  model_b: xxxx

context_windows:
  model_a: 2048
  model_b: 4096

intelligence_rank:
  - model_a
  - model_b

cost_rank:
  - model_a
  - model_b
```

A.8 TOOL CONFIGURATION

Web Search. Tool Card A.1 specifies the web-search interface used in our experiments. It accepts a query string and returns the top five results from the Bing Search API. During training, we use ZeroSearch (Sun et al., 2025) to simulate search results and reduce API cost.

Tool Card A.1: Web Search

```
=== Name ===

WebSearch

=== Description ===

Retrieve the top five results for the given query through the
web search.

=== Parameters ===
```

```
{
    "type": "object",
    "properties": {
        "query": {
            "type": "string",
            "description": "Query string. Include a single search
            ↪  query."
        }
    },
    "required": ["query"]
}
```

Python Code Interpreter. Tool Card A.2 specifies the Python-execution interface. We use SandboxFusion (bytedance, 2024) as the sandboxed execution backend.

Tool Card A.2: Python Code Interpreter

```
=== Name ===

PythonInterpreter

=== Description ===

Execute Python code in a sandboxed environment.  Use this to
run Python code and get the execution results.

\textbf {Make sure to use print() for any output you want to
see in the results.}

=== Parameters ===
```

```
{
    "type": "object",
    "properties": {
        "code": {
            "type": "string",
            "description": "The Python code to execute. Remember to
            ↪  use print() statements for any output you want to
            ↪  see."
        }
    },
    "required": ["code"]
}
```

File Parser. Tool Card A.3 specifies the file-parsing interface, which accepts a file path and returns the parsed content.

Tool Card A.3: File Parser

```
=== Name ===

FileParser

=== Description ===

Parse a file and return its content.  Supports:  text (.txt,
.md, .py), spreadsheets (.xlsx, .csv), documents (.pdf, .docx,
.pptx), data files (.json, .jsonl, .xml, .pdb), archives
(.zip), and audio files with transcription (.mp3, .wav, .flac,
.m4a, .ogg).

=== Parameters ===
```

```
{
    "type": "object",
    "properties": {
        "file_name": {
            "type": "string",
            "description": "Name of the file to parse (relative to
            ↪  the file base directory).",
            "examples": ["report.pdf", "data.csv", "audio.mp3"]
        }
    },
    "required": ["file_name"]
}
```

Image Question Answering. Tool Card A.4 specifies the visual question-answering interface. It accepts an image and a question and uses Qwen2.5-VL-32B-Instruct (Bai et al., 2025) to produce the answer.

Tool Card A.4: Image Question Answering

```
=== Name ===

ImageQuestionAnswering

=== Description ===

Answer the question based on the image.  Help with image
understanding, object recognition, scene analysis, and
answering questions about visual content.

=== Parameters ===
```

```
{
    "type": "object",
    "properties": {
        "image_name": {
            "type": "string",
            "description": "Image name to be analyzed."
        },
        "question": {
            "type": "string",
```

```
            "description": "A clear, concise question about the
            ↪  image's content. For best results, ask
            ↪  straightforward factual questions rather than
            ↪  complex or multi-step reasoning questions."
        }
    },
    "required": ["image_name", "question"]
}
```

Environment-Specific Tools for ALFWorld. Tool Cards A.5–A.10 specify the interfaces used to interact with ALFWorld (Shridhar et al., 2021).

Tool Card A.5: ALFWorld Goto

```
=== Name ===

goto

=== Description ===

Navigate to a specific receptacle or location in the
environment.  Move you to the target location and interact
with objects there.  You must be at the receptacle's location
to interact with it.  After successfully going to a location,
'You arrive at [location]' confirmation will be shown.

=== Parameters ===
```

```
{
    "type": "object",
    "properties": {
        "recep": {
            "type": "string",
            "description": "Name of the target receptacle or
            ↪  location (e.g., 'cabinet 1', 'fridge 1',
            ↪  'countertop 2')."
        }
    },
    "required": ["recep"]
}
```

Tool Card A.6: ALFWorld Open

```
=== Name ===

open

=== Description ===

Open a closed receptacle to access its contents.  Is necessary
before you can see or interact with objects inside containers
like cabinets, drawers, fridge, microwave, or safe.  You must
be at the receptacle's location to open it.  After
successfully opening a receptacle, 'The [receptacle] is open'
confirmation will be shown.

=== Parameters ===
```

```
{
    "type": "object",
```

```
    "properties": {
        "recep": {
            "type": "string",
            "description": "Name of the receptacle to open (e.g.,
            ↪  'cabinet 2', 'fridge 1', 'microwave 1')."
        }
    },
    "required": ["recep"]
}
```

Tool Card A.7: ALFWorld Use

```
=== Name ===

use

=== Description ===

Use an object or receptacle.  You must be at the object's
location to use it.  The commonly used object is desklamps for
lighting objects.  After successfully using an desklamp, 'You
turn on [object]' confirmation will be shown.

=== Parameters ===
```

```
{
    "type": "object",
    "properties": {
        "obj": {
            "type": "string",
            "description": "Name of the object or receptacle to use
            ↪  (e.g., 'desklamp 1')."
        }
    },
    "required": ["obj"]
}
```

Tool Card A.8: ALFWorld Take

```
=== Name ===

take

=== Description ===

Pick up an object from a receptacle or surface.  Grab items
like food, utensils, containers, and other objects from their
current location.  Make sure the receptacle is open if the
object is inside (cabinets, drawers, fridge, microwave, safe).
Only hold one object at a time - must move the current object
before taking another.  After successfully taking an object,
'You pick up [object]' confirmation will be shown.

=== Parameters ===
```

```
{
    "type": "object",
    "properties": {
        "obj": {
            "type": "string",
```

```
            "description": "Name of the object to pick up (e.g.,
            ↪  'apple 1', 'mug 2', 'knife 1')."
        },
        "from": {
            "type": "string",
            "description": "Name of the source receptacle or
            ↪  surface (e.g., 'countertop 1', 'cabinet 4')."
        }
    },
    "required": ["obj", "from"]
}
```

Tool Card A.9: ALFWorld Clean

```
=== Name ===

clean

=== Description ===

Clean an object using a cleaning receptacle.  Wash or clean
items like dishes, utensils, or food items.  The most common
cleaning receptacle is the sinkbasin.  Must have the object
in the inventory first (use 'take').  After successfully
cleaning an object, 'You clean [object] using [receptacle]'
confirmation will be shown.

=== Parameters ===
```

```
{
    "type": "object",
    "properties": {
        "obj": {
            "type": "string",
            "description": "Name of the object to clean (e.g., 'mug
            ↪  1', 'knife 1')."
        },
        "with": {
            "type": "string",
            "description": "Receptacle used for cleaning (e.g.,
            ↪  'sinkbasin 1')."
        }
    },
    "required": ["obj", "with"]
}
```

Tool Card A.10: ALFWorld Heat

```
=== Name ===

heat

=== Description ===
Heat an object using a heating receptacle.  Warm up items
like food or containers to make them hot.  Must have the
object in the inventory first (use 'take').  The microwave is
the primary heating location.  If get 'Nothing happens', make
sure at the microwave location and have the object in the
inventory.  After successfully heating an object, 'You heat
[object] using [receptacle]' confirmation will be shown.
```

```
=== Parameters ===
```

```
{
    "type": "object",
    "properties": {
        "obj": {
            "type": "string",
            "description": "Name of the object to heat (e.g.,
            ↪  'apple 1', 'mug 1')."
        },
        "with": {
            "type": "string",
            "description": "Receptacle used for heating (e.g.,
            ↪  'microwave 1')."
        }
    },
    "required": ["obj", "with"]
}
```

Environment-Specific Tools for WebShop. Tool Cards A.11–A.12 specify the interfaces used to interact with WebShop (Yao et al., 2022).

Tool Card A.11: WebShop Search

```
=== Name ===
```

```
search
```

```
=== Description ===
```

```
Search for products with the given query.  Use this only if a
[Search] button appears in the tool call result.  Note:  Click
the [Back to Search] button if there's no [Search] button via
the 'click' tool, and then try again with a more specific
query.
```

```
=== Parameters ===
```

```
{
    "type": "object",
    "properties": {
        "query": {
            "type": "string",
            "description": "Search query for products"
        }
    },
    "required": ["query"]
}
```

```
|
```

Tool Card A.12: WebShop Click

```
=== Name ===
```

```
click
```

```
=== Description ===
```

```
Click on a button or product.  Use this only if a [button] is
present in the tool call result.  Click the [Buy Now] button
on its product page to finish the shopping task if the most
suitable product is identified.
=== Parameters ===
```

```
{
    "type": "object",
    "properties": {
        "button": {
            "type": "string",
            "description": "Name of the button or product ASIN,
            ↪  don't add '[]' around the button name"
        }
    },
    "required": ["button"]
}
```

A.9 Dataset Details

We summarize the data sources used for training and evaluation in this section.

A.9.1 Training Data

DAPO-Math. DAPO-Math (Yu et al., 2025) is a curated mathematical-reasoning dataset introduced with the DAPO reinforcement-learning system. Its commonly used DAPO-Math-17K split contains 17K competition-style prompts with integer answers, supporting verifiable answer-level rewards without requiring reference solution traces.

OpenScienceReasoning-2. OpenScienceReasoning-2 (NVIDIA, 2025) contains synthetic reasoning examples across scientific and academic domains. We include it to expose the orchestrator to knowledge-intensive tasks beyond mathematical problem solving.

Natural Questions. Natural Questions (Kwiatkowski et al., 2019) contains more than 300K training examples constructed from real information-seeking queries, with long- and short-answer annotations derived from Wikipedia pages. We use it to train retrieval-oriented orchestration and evidence integration.

HotpotQA. HotpotQA (Yang et al., 2018) contains approximately 113K multi-hop questions with supporting-fact annotations across multiple Wikipedia documents. It provides training examples for decomposing evidence-gathering tasks and integrating information across sources.

WebDancer. WebDancer (Wu et al., 2025) provides data for agentic information seeking that involves iterative browsing, evidence collection, and answer synthesis. We use these examples to train long-horizon web-research behavior.

ALFWorld and WebShop. We additionally include instances from ALFWorld (Shridhar et al., 2021) and WebShop (Yao et al., 2022) to train sequential environment interaction and tool-mediated decision making.

A.9.2 Evaluation Benchmarks

AIME24 and AIME25. AIME24 and AIME25 comprise problems from the 2024 and 2025 American Invitational Mathematics Examinations. Each problem has a three-digit integer answer in the range 000–999 and requires multi-step competition-level reasoning, enabling automatic answer verification.

MATH500. MATH500 (Hendrycks et al., 2021) is a 500-problem subset of MATH spanning algebra, number theory, geometry, counting and probability, prealgebra, and precalculus. We use it to evaluate mathematical reasoning across a broader range of topics and difficulty levels than AIME.

ALFWorld. ALFWorld (Shridhar et al., 2021) is a text-based interactive benchmark aligned with the embodied ALFRED environment. It covers six household task types requiring long-horizon planning, environment grounding, and execution through actions such as navigation, object manipulation, heating, cooling, and cleaning.

WebShop. WebShop (Yao et al., 2022) is a simulated e-commerce environment in which an agent must satisfy a natural-language shopping request through search, product inspection, attribute comparison, and purchase actions. It evaluates grounded sequential decision making under user-specified constraints.

GAIA. GAIA (Mialon et al., 2024) evaluates general assistants on real-world questions requiring combinations of reasoning, browsing, tool use, file manipulation, and multimodal understanding. We use it to assess long-horizon information gathering and evidence integration.

HLE. Humanity's Last Exam (HLE) (Phan et al., 2025) is a challenging multimodal benchmark spanning mathematics, science, the humanities, and other specialized domains. Its expert-written, closed-ended questions evaluate broad-domain knowledge and precise answer generation.

A.10 Baseline Details

We group the evaluated methods into single-agent approaches, multi-agent collaboration frameworks, and agentic orchestration methods. Unless otherwise stated, the baselines use the common executor backbones and tool interfaces described in the main text.

Direct Reasoning. This baseline prompts a single executor to solve each task directly. It does not use iterative environment interaction, multi-agent collaboration, or a learned orchestration policy. In contrast, E^3-Orch explicitly orchestrates multiple executors through iterative milestone decomposition, dependency-aware planning, and adaptive executor assignment.

ReAct. ReAct (Yao et al., 2023) interleaves natural-language reasoning, actions, and environment observations. We use it as an interactive single-agent baseline with access to the same task-specific tools, but without explicit coordination among multiple executors. Unlike ReAct, which makes reasoning and tool-use decisions within a single-agent trajectory, E^3-Orch additionally learns how to decompose tasks across milestones, construct dependency-aware execution plans, and allocate subtasks among heterogeneous executors.

Reflexion. Reflexion (Shinn et al., 2023) uses evaluator feedback to produce verbal reflections that condition subsequent attempts. It provides inference-time self-improvement through episodic memory without updating model parameters. In contrast, E^3-Orch uses execution feedback during reinforcement learning to optimize an explicit orchestration policy, with feedback assigned to intermediate milestone and plan decisions rather than used solely as textual context for subsequent attempts.

AutoGen. AutoGen (Wu et al., 2023) supports task solving through role-based multi-agent conversations. We use its default two-agent configuration, in which an assistant performs reasoning and response generation while a user-proxy agent mediates requests and, when needed, tool or code execution. Whereas AutoGen coordinates agents through a predefined conversational structure, E^3-Orch learns an execution-aware orchestration policy that dynamically determines milestones, subtask dependencies, and executor assignments as execution progresses.

Multi-Agent Debate. Multi-Agent Debate (Du et al., 2024) elicits candidate solutions from multiple agents, allows them to exchange critiques, and aggregates their reasoning into a final answer. The interaction follows a fixed debate protocol rather than a profile-conditioned runtime orchestration policy. In contrast, E^3-Orch does not prescribe a fixed communication pattern. It dynamically constructs dependency-aware execution graphs and delegates different subtasks according to the current execution history and executor profiles.

MAS-ZERO. MAS-ZERO (Ke et al., 2025) is a training-free method that constructs a problem-specific multi-agent configuration at inference time. A meta-agent decomposes the task, critiques intermediate solutions, and refines the agent composition and workflow without relying on a validation

set. Both MAS-ZERO and E^3-ORCH adapt multi-agent structures to individual tasks, but they differ in how orchestration is obtained: MAS-ZERO performs training-free inference-time construction, whereas E^3-ORCH learns an orchestration policy from execution feedback and organizes execution through iterative milestone-level plans with explicit decision-level credit assignment.

DAAO. Difficulty-Aware Agentic Orchestration (DAAO) (Su et al., 2026) estimates query difficulty and adapts workflow depth, operators, and model assignments accordingly. It provides query-specific workflow construction through a difficulty-aware inference-time controller. In contrast, E^3-ORCH conditions orchestration not only on the initial query but also on accumulated execution outcomes, allowing subsequent milestones and plans to evolve with the task state. Its orchestration policy is further optimized through reinforcement learning rather than determined solely by an inference-time controller.

AGENTFLOW. AGENTFLOW (Li et al., 2025) coordinates a planner, executor, verifier, and generator through shared evolving memory. It trains the planner within this iterative interaction loop using trajectory-level feedback. Similar to E^3-ORCH, AGENTFLOW learns orchestration through reinforcement learning and supports iterative execution. The key difference is the optimization granularity: E^3-ORCH explicitly structures orchestration into milestone and plan decisions and constructs tree-structured rollouts from shared execution histories, enabling decision-level performance and cost feedback rather than assigning the same trajectory-level signal across the entire interaction.

PUPPETEER. PUPPETEER (Dang et al., 2025) trains a centralized orchestrator to sequence and prioritize agents as the task state evolves. The orchestration policy is optimized from execution feedback rather than being specified entirely through a fixed prompt or workflow. Both PUPPETEER and E^3-ORCH therefore learn adaptive orchestration policies, but E^3-ORCH additionally introduces the milestone-plan-act abstraction, represents each milestone plan as an explicit dependency graph that supports parallel execution, and performs decision-level optimization through tree-structured rollouts and multi-objective rewards.

MAS-ORCHESTRA. MAS-ORCHESTRA (Ke et al., 2026) formulates holistic multi-agent construction as a function-calling reinforcement-learning problem. Its orchestrator jointly determines task decomposition, agent selection, dependencies, and information flow in a single orchestration decision. E^3-ORCH optimizes a closely related set of orchestration dimensions but distributes them across iterative milestone and plan decisions: each milestone locally scopes decomposition and dependency-aware execution, while subsequent milestones are conditioned on intermediate outcomes. This iterative abstraction further provides intermediate decision points for tree-structured credit assignment.

MASROUTER. MASROUTER (Yue et al., 2025) uses cascaded controllers to determine the collaboration mode, allocate agent roles, and assign an LLM to each role. It therefore adapts both the multi-agent configuration and the underlying model allocation to each query. In contrast, E^3-ORCH treats executor selection as one component of a broader orchestration problem that also includes iterative milestone decomposition, dependency-aware planning, and execution scheduling, and learns these decisions jointly through a reinforcement-learned LLM orchestrator conditioned on executor profiles.

A.11 BASELINES IMPLEMENTATION DETAILS

For single-agent baselines, we instantiate the workflow with a single executor, `model_b`. For multi-agent collaboration baselines, we instantiate it with all executors in $\mathcal{A}$. For the agentic orchestration baselines, we replace only the method's own model pool with $\mathcal{A}$, leaving its remaining components, such as its role pool, unchanged. For all baselines, we ensure the use of the same orchestrator backbones and tool interfaces as in E^3-ORCH.

A.12 TRAINING PARAMETERS

Table 3 summarizes the hyperparameters used to optimize the orchestrator in E^3-ORCH.

Table 3: Training hyperparameters for the orchestrator.

Parameter	Value
Batch Size	64
Initial Trajectories (B)	2
Branches per Expansion (N)	4
Expansion Rounds (E)	2
Rollout Temperature	0.7
Rollout Top-p	0.8
Rollout Top-k	20
Maximum Number of Generation Tokens	2048
Learning Rate	1×10^{-6}
KL Loss Coefficient	1×10^{-4}
ϵ_l	0.2
ϵ_r	0.28
Gradient Clipping	1.0
Number of Training Steps	150
λ_{perf}	1.0
λ_{cost}	0.1
λ_{comp}	0.1

A.13 Definition of Intelligence Efficiency

Cost of a Single Run. For one evaluation instance q, let $t_k(q)$ denote the number of output tokens generated by executor a_k over all subtasks delegated to it while solving q, summed across every milestone and plan of the trajectory. We define the *cost-weighted executor output tokens* of that instance as

$$T(q) = \sum_{k=1}^{K} \kappa_k \, t_k(q), \tag{6}$$

where κ_k is the relative cost of executor a_k declared in its operational profile ϕ_k (§ 3.1). We set κ_k to the executor's relative configured cost normalized by that of the cheapest executor in the pool, so that the cheapest executor has $\kappa_k = 1$ and $T(q)$ is expressed in units of tokens of the cheapest executor. In our two-executor configuration this gives $\kappa = (1.0,\ 2.0)$ for `model_a` and `model_b`, and $(1.0,\ 2.0,\ 3.0)$ in the three-executor configuration of § B.3. We report T in thousands of tokens (K) and, unless stated otherwise, averaged over evaluation instances and over the repeated runs.

Intelligence Efficiency. Let P denote the task performance of a method on a benchmark, in the units reported in Table 1, and let T denote its mean cost-weighted executor output tokens per instance. Intelligence efficiency is their ratio,

$$\eta = \frac{P}{T}, \tag{7}$$

measured in performance points per 1K cost-weighted executor output tokens. It answers how much task-solving capability a method extracts from a fixed execution budget, rather than how much capability it reaches at unbounded budget. Its reciprocal $1/\eta = T/P$ is the expected cost-weighted token spend per *solved* instance, which is the quantity we report when comparing token spend at matched capability. The two summarize the same measurement, and we use whichever is more direct for the claim being made.

Relation to the Cost Reward. The quantity $C(\nu)$ optimized by the cost reward in Equation (2) is the same cost-weighted output-token count of Equation (6), restricted to the subtree rooted at decision ν. The training signal is a within-sibling z-score of $C(\nu)$, which is comparable only among alternatives sharing an execution history, whereas η is an evaluation-time aggregate over complete trajectories. Training therefore optimizes the numerator and denominator of η through separate rewards, and η is used only for reporting.

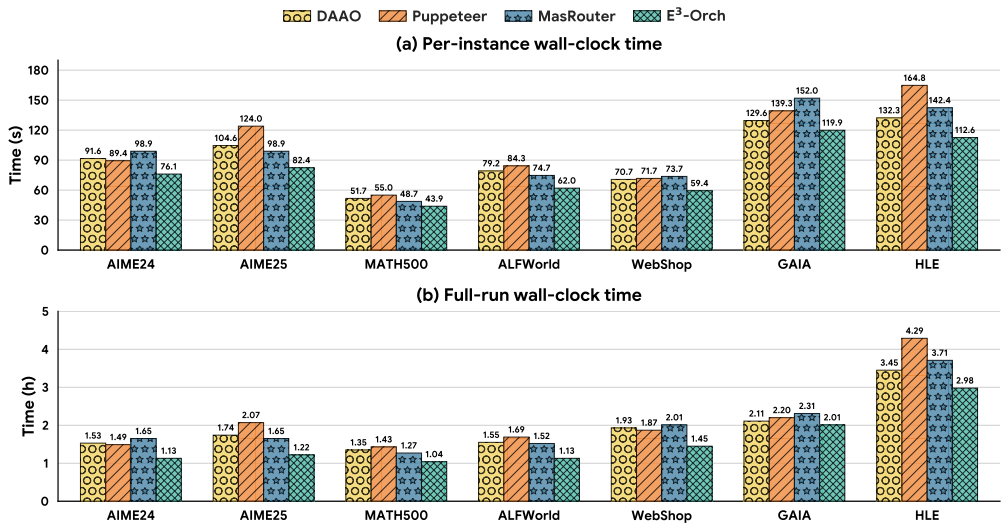

Figure 8: Wall-clock time comparison of E^3-ORCH and baseline methods on evaluation benchmarks.

B ADDITIONAL EXPERIMENTAL RESULTS AND ANALYSIS

B.1 SCALING THE ORCHESTRATOR

Table 4 compares Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct as orchestrator backbones for E^3-ORCH. Scaling the orchestrator from 3B to 7B improves performance on every reported benchmark, which suggests that the learned orchestration policy benefits from a stronger backbone.

Table 4: Performance of E^3-ORCH with 3B and 7B orchestrator backbones.

Method	Orchestrator Size	Mathematical		Embodied		Deep Research	
		AIME24/25	MATH500	ALFWorld	WebShop	GAIA	HLE
E^3-ORCH	3B	$32.3_{\pm 0.5}$/$28.7_{\pm 0.4}$	$85.4_{\pm 0.4}$	$66.7_{\pm 0.5}$	$58.0_{\pm 0.6}$	$15.2_{\pm 0.4}$	$5.1_{\pm 0.3}$
E^3-ORCH	7B	$35.0_{\pm 0.4}$/$31.5_{\pm 0.5}$	$90.4_{\pm 0.4}$	$70.2_{\pm 0.5}$	$62.7_{\pm 0.6}$	$20.5_{\pm 0.4}$	$7.8_{\pm 0.3}$

B.2 WALL-CLOCK TIME ANALYSIS

Evaluation Wall-Clock Time. To demonstrate the efficiency of E^3-ORCH, we also measure its wall-clock time on evaluation benchmarks and compare it against the baseline methods. As shown in Figure 8, E^3-ORCH consistently achieves lower wall-clock times compared to the baseline methods across all evaluation benchmarks regarding per-instance and full-benchmark measurements.

Training Wall-Clock Time. We further compare the trained methods by the wall-clock time required to complete reinforcement learning, measured end to end under identical hardware and software settings. Tree-structured rollouts expand a branch from an existing decision node rather than re-executing a trajectory from scratch, so each additional candidate costs only the execution below its branch point. As shown in Table 5, E^3-ORCH therefore completes training faster than AGENTFLOW, which performs a full rollout for every training instance. Router-based methods such as MASROUTER remain cheaper to train, since they do not require the RLVR process, but they do not match E^3-ORCH on task performance or on robustness to changes in the executor pool (§ B.3).

Table 5: Training wall-clock time of E^3-ORCH and the strongest trained baseline.

Method	Time (h)
E^3-ORCH	36.2
AGENTFLOW	40.6

B.3 Generalization Across Executor Configurations

We evaluate whether the profile-conditioned interface transfers to executor configurations that differ from the two-executor setup in § A.7. Both tables report task performance and token cost, their ratio is the intelligence efficiency of Figure 4.

Generalization to Unseen Executor Configurations. We first expand the pool to three executors, adding a larger Qwen2.5-14B-Instruct as `model_c`. A larger pool is not free: the added executor is the most capable but also the most expensive one, so a policy that simply reaches for it converts the extra capability into extra cost. Table 6 shows that E^3-ORCH does not. It attains the best task performance on all seven benchmarks while also spending the fewest tokens, 12–37% below the cheapest baseline in each column, for 1.46–2.92× the intelligence efficiency of the stronger baseline. MASROUTER shows the opposite pattern: its cascaded controllers must now route over a larger action space, and its performance falls well below its two-executor result while its token cost stays above E^3-ORCH's on every benchmark. Extending the pool therefore tests not only whether an orchestrator still functions, but whether it can decline the more expensive option when the task does not need it.

```yaml
model_list:
  - model_a
  - model_b
  - model_c

model_names:
  model_a: Qwen/Qwen2.5-7B-Instruct
  model_b: Qwen/Qwen2.5-7B-Instruct
  model_c: Qwen/Qwen2.5-14B-Instruct

model_urls:
  model_a: xxxx
  model_b: xxxx
  model_c: xxxx

context_windows:
  model_a: 512
  model_b: 2048
  model_c: 2048

intelligence_rank:
  - model_a
  - model_b
  - model_c

cost_rank:
  - model_a
  - model_b
  - model_c
```

Table 6: Task performance (↑) and token cost (↓, weighted executor output tokens per instance in thousands) under a three-executor configuration.

Method	Metric	Mathematical		Embodied		Deep Research	
		AIME24/25	MATH500	ALFWorld	WebShop	GAIA	HLE
AGENTFLOW	Performance	$34.0_{\pm 0.5}$/$28.8_{\pm 0.4}$	$89.1_{\pm 0.4}$	$66.5_{\pm 0.5}$	$58.5_{\pm 0.6}$	$17.4_{\pm 0.5}$	$5.9_{\pm 0.4}$
	Tokens (K)	$11.8_{\pm 1.0}$/$12.9_{\pm 1.3}$	$5.8_{\pm 0.6}$	$4.7_{\pm 0.4}$	$5.0_{\pm 0.5}$	$23.6_{\pm 5.1}$	$15.3_{\pm 2.0}$
MASROUTER	Performance	$20.0_{\pm 0.4}$/$16.8_{\pm 0.3}$	$83.5_{\pm 0.5}$	$52.0_{\pm 0.6}$	$40.4_{\pm 0.4}$	$8.5_{\pm 0.3}$	$3.7_{\pm 0.3}$
	Tokens (K)	$8.8_{\pm 0.9}$/$12.3_{\pm 1.1}$	$5.2_{\pm 0.7}$	$5.0_{\pm 0.5}$	$5.8_{\pm 0.5}$	$15.3_{\pm 2.0}$	$12.9_{\pm 1.3}$
E^3-ORCH	Performance	**$37.2_{\pm 0.4}$/$33.5_{\pm 0.5}$**	**$92.1_{\pm 0.3}$**	**$73.3_{\pm 0.5}$**	**$64.9_{\pm 0.4}$**	**$23.1_{\pm 0.6}$**	**$9.6_{\pm 0.3}$**
	Tokens (K)	**$7.8_{\pm 0.2}$/$8.5_{\pm 0.5}$**	**$4.0_{\pm 0.4}$**	**$3.0_{\pm 0.3}$**	**$3.3_{\pm 0.4}$**	**$10.4_{\pm 0.8}$**	**$10.4_{\pm 0.8}$**

Generalization to Enriched Executor Descriptions. We next enrich each executor profile with rankings of domain-specific capabilities, replacing `model_b` with a mathematics-specialized executor and exposing separate general and mathematical intelligence rankings alongside the cost ranking. This changes the schema the orchestrator reads, not just the pool it reads about. In Table 7, E^3-ORCH again attains the best task performance and the lowest token cost on all seven benchmarks, spending 17–40% fewer tokens than the cheapest baseline and reaching 1.61–2.63× its intelligence efficiency, without any retraining for the revised schema. Its margin over the baselines does not shrink under either extension, which is what the profile-conditioned interface is intended to provide: the orchestrator reads capability and cost from Φ rather than from a fixed executor set, so a richer or larger pool changes its input rather than its interface.

```yaml
model_list:
  - model_a
  - model_b

model_names:
  model_a: Qwen/Qwen2.5-7B-Instruct
  model_b: Qwen/Qwen2.5-Math-7B-Instruct

model_urls:
  model_a: xxxx
  model_b: xxxx

context_windows:
  model_a: 2048
  model_b: 2048

general_intelligence_rank:
  - model_b
  - model_a

mathematical_intelligence_rank:
  - model_a
  - model_b

cost_rank:
  - model_a
  - model_b
```

Table 7: Task performance (↑) and token cost (↓, weighted executor output tokens per instance in thousands) with enriched, domain-specific executor profiles.

Method	Metric	Mathematical		Embodied		Deep Research	
		AIME24/25	MATH500	ALFWorld	WebShop	GAIA	HLE
AGENTFLOW	Performance	$32.8_{\pm 0.4}$/$27.0_{\pm 0.5}$	$87.9_{\pm 0.5}$	$65.6_{\pm 0.4}$	$57.4_{\pm 0.6}$	$15.6_{\pm 0.5}$	$5.1_{\pm 0.3}$
	Tokens (K)	$12.3_{\pm 1.1}$/$12.3_{\pm 1.1}$	$5.8_{\pm 0.6}$	$4.8_{\pm 0.4}$	$5.1_{\pm 0.6}$	$20.0_{\pm 2.6}$	$15.3_{\pm 2.0}$
MASROUTER	Performance	$18.8_{\pm 0.5}$/$15.1_{\pm 0.3}$	$82.7_{\pm 0.6}$	$51.2_{\pm 0.5}$	$38.8_{\pm 0.4}$	$7.7_{\pm 0.4}$	$2.7_{\pm 0.3}$
	Tokens (K)	$9.7_{\pm 1.0}$/$11.8_{\pm 1.0}$	$5.5_{\pm 0.6}$	$5.2_{\pm 0.7}$	$6.2_{\pm 0.5}$	$13.6_{\pm 1.5}$	$12.3_{\pm 1.1}$
E^3-ORCH	Performance	**$35.2_{\pm 0.5}$**/**$32.8_{\pm 0.4}$**	**$91.3_{\pm 0.4}$**	**$71.0_{\pm 0.5}$**	**$63.8_{\pm 0.5}$**	**$21.4_{\pm 0.6}$**	**$8.5_{\pm 0.4}$**
	Tokens (K)	**$7.5_{\pm 0.7}$**/**$9.1_{\pm 0.6}$**	**$3.8_{\pm 0.4}$**	**$2.9_{\pm 0.3}$**	**$3.1_{\pm 0.4}$**	**$11.3_{\pm 0.5}$**	**$9.7_{\pm 1.0}$**

B.4 ANALYSIS OF ORCHESTRATION ABSTRACTION

To isolate the contribution of the milestone abstraction itself from the learning machinery built on top of it, we remove milestones and instead treat the entire query q as a single milestone. Under this variant, the orchestrator generates a single plan for the complete task, tree branching occurs only at plan decisions, and all other training and execution settings remain identical to E^3-ORCH. This setup directly instantiates the "plan a complete workflow upfront" regime discussed in § 1, allowing us to evaluate the effect of the milestone abstraction in isolation.

As shown in Table 8, removing milestones produces the largest overall degradation among the ablations we consider. Task performance decreases by 6.0, 5.8, and 3.9 points on mathematical reasoning, embodied decision-making, and deep research, respectively. These are the largest drops among all variants in Table 2 for mathematical reasoning and embodied decision-making, and the second largest for deep research. At the same time, token cost increases by 37%, 85%, and 27%, reducing intelligence efficiency to 0.65, 0.49, and 0.57× that of E^3-ORCH. The resulting efficiency is lower than that of any other ablation on embodied decision-making and deep research. Notably, removing the milestone abstraction is more detrimental than replacing tree-structured rollouts or removing any individual reward, indicating that the improvements in § 4.2 cannot be attributed solely to applying the proposed reinforcement learning procedure at an arbitrary decision granularity.

The domain-specific effects further illustrate the role of milestones. Deep research exhibits the largest *relative* performance degradation, at 27%, consistent with the motivation that committing to a complete workflow upfront makes it harder to revise the execution strategy when intermediate results reveal new information requirements. Embodied decision-making exhibits the largest increase in token cost, consistent with the benefit of periodically re-scoping execution at milestone boundaries rather than continuing a long-horizon plan after the task state has changed. Finally, removing milestones eliminates milestone-level branching and leaves only plan-level alternatives, thereby reducing the set of decision points at which matched downstream outcomes can be compared. Credit must consequently be assigned over longer spans of execution, directly exacerbating the decision-level credit-assignment challenge that motivates the milestone abstraction.

Table 8: Ablation results of orchestration abstraction, where we remove the milestone structure. We report task performance (↑) and token cost (↓), the weighted executor output tokens per instance in thousands, their ratio is the intelligence efficiency of Figure 4.

Method	Mathematical		Embodied		Deep Research	
	Perf.	Tokens (K)	Perf.	Tokens (K)	Perf.	Tokens (K)
E^3-ORCH	**52.3**$_{\pm 0.4}$	**6.2**$_{\pm 0.5}$	**66.5**$_{\pm 0.5}$	**3.3**$_{\pm 0.4}$	**14.2**$_{\pm 0.4}$	**10.4**$_{\pm 0.4}$
w/o Milestones	46.3$_{\pm 0.4}$	8.5$_{\pm 0.7}$	60.7$_{\pm 0.5}$	6.1$_{\pm 0.4}$	10.3$_{\pm 0.3}$	13.2$_{\pm 0.7}$

B.5 ABLATION OF COMPLETENESS REWARD

The completeness reward is the only component that directly targets the structural validity of an orchestration decision rather than its downstream outcome. We therefore evaluate its effect through plan validity rather than task performance. Following the definition in § 3.3.2, we consider four criteria: whether a plan is successfully parsed with all required fields, whether its dependencies form a directed acyclic graph (DAG), whether every step is assigned to a valid executor in the pool, and whether the resulting plan can be executed to produce a milestone-level outcome.

Table 9: Ablation of the completeness reward. We report the percentage of orchestration decisions passing each validity check, measured over all plans generated during evaluation on all benchmarks. Format and assignment validity are syntactic checks, DAG validity is structural, and **Executable Plan** requires all three together with successful production of a milestone-level outcome.

Method	Valid Format ↑	Valid DAG ↑	Valid Assignment ↑	Executable Plan ↑
E^3-ORCH	100.0	96.3	100.0	95.4
w/o Completeness Reward	74.3	62.1	85.7	60.4

As shown in Table 9, E^3-ORCH achieves perfect format and executor-assignment validity, while its remaining failures arise primarily from malformed dependency structures, with 3.7% of plans violating DAG validity. Removing the completeness reward degrades all four criteria, but to markedly different extents. Executor-assignment validity decreases by 14.3 percentage points, whereas DAG validity and overall executability decrease by 34.2 and 35.0 points, respectively. This difference reflects the nature of the underlying constraints: selecting a valid executor is largely a local decision, facilitated by explicitly enumerating the admissible executor identifiers in the prompt, whereas constructing an acyclic dependency graph imposes a global structural constraint over all steps in the

plan. The ablation results therefore indicate that prompting alone is insufficient to reliably enforce such global structural consistency.

The impact of completeness extends beyond output validity. A non-executable plan produces no milestone-level outcome and consequently provides no downstream signal from either the performance or cost reward. With only 60.4% of plans executable after removing the completeness reward, roughly two in five orchestration decisions fail to yield usable downstream feedback. This also reduces the number of valid sibling branches available for the within-sibling comparisons underlying the other two rewards. The completeness reward thus serves as a structural safeguard for decision-level learning, helping preserve the valid rollout branches required by the performance and cost objectives. Notably, the completeness reward remains effective with a relatively small weight ($\lambda_{\text{comp}} = 0.1$), while removing it entirely leads to substantially more invalid orchestration decisions.

B.6 Case Studies

Performance-Oriented Orchestration. The paired AIME25 examples in Case B.1 and Case B.2 illustrate the role of milestone decomposition and dependency-aware planning. E^3-ORCH decomposes the problem into three verifiable milestones—deriving the divisibility constraints, counting valid assignments, and computing the requested difference—and specifies an executor and dependency list for every step. The resulting plan forms an executable DAG. By contrast, the baseline trace contains a missing dependency field and a circular dependency, making the plan unreliable to execute. In this pair of traces, the structured milestone-plan-act interface avoids the dependency and format errors present in the baseline plan. We draw the comparison from these two examples alone and do not quantify how often such errors occur.

Efficiency-Oriented Orchestration. The paired GAIA examples in Case B.3 and Case B.4 illustrate capability–cost-aware executor assignment. E^3-ORCH delegates straightforward retrieval and extraction to the lower-cost `model_a`, while using the stronger `model_b` for the consistency check needed to avoid an entity mismatch. The baseline trace reaches the same final answer but repeatedly invokes `model_b` for simple or redundant checks and adds an unnecessary summarization step. This contrast is consistent with the intended behavior of the learned policy: allocate stronger executors only when their additional capability is useful, and otherwise favor lower-cost execution.

Case B.1: E^3-ORCH Case on AIME25

Task. There are $8! = 40320$ eight-digit positive integers that use each of the digits $1, 2, 3, 4, 5, 6, 7, 8$ exactly once. Let N be the number of these integers that are divisible by 22. Find the difference between N and 2025.

Trace. 3 milestones; 10 planned steps. **Assignments.** `model_a`: 7; `model_b`: 3.

Milestone 1. Derive the divisibility constraints for 22.

Step	Sub-task	Executor	Depends on
1	Factor 22 into 2×11 and identify the required divisibility conditions.	`model_a`	$\emptyset$
2	Apply the divisibility rule for 11 to the digit positions.	`model_b`	[1]
3	Verify that the sums of digits in odd and even positions must be equal, and hence both must be 18.	`model_a`	[2]
4	Verify that the last digit must be even to satisfy divisibility by 2.	`model_a`	[1]

Outcome. A valid number must satisfy two constraints: the odd-position digits and even-position digits each sum to 18, and the last digit is even.

Milestone 2. Count the valid digit assignments satisfying the derived constraints.

Step	Sub-task	Executor	Depends on
1	Enumerate all 4-digit subsets of $\{1,2,3,4,5,6,7,8\}$ whose sum is 18.	`model_b`	$\emptyset$
2	Count the number of arrangements of each valid subset in the odd positions.	`model_a`	[1]
3	Count the number of valid arrangements of the complementary digits in the even positions under the constraint that the last digit is even.	`model_a`	[1]
4	Aggregate the counts from odd-position and even-position arrangements to obtain N.	`model_b`	[2, 3]

Outcome. There are 8 valid choices for the odd-position digit set. For each choice, there are $4!$ arrangements of the odd positions and $2 \times 3!$ valid arrangements of the even positions. Thus,

$$N = 8 \times 4! \times 2 \times 3! = 2304.$$

Milestone 3. Compute the final requested difference.

Step	Sub-task	Executor	Depends on
1	Compute $2304 - 2025$.	`model_a`	$\emptyset$
2	Return the final answer.	`model_a`	[1]

Final answer. $\boxed{279}$

Analysis

The trace separates constraint derivation, counting, and final arithmetic into three milestones. Executor `model_b` handles the divisibility argument, subset enumeration, and count aggregation, while `model_a` performs local checks and arithmetic. Explicit dependencies expose independent checks and keep the counting steps ordered. The recorded final answer is 279.

Case B.2: Baseline Case on AIME25

Task. There are $8! = 40320$ eight-digit positive integers that use each of the digits $1, 2, 3, 4, 5, 6, 7, 8$ exactly once. Let N be the number of these integers that are divisible by 22. Find the difference between N and 2025.

Trace. 1 milestone; 5 planned steps. **Assignments.** `model_a`: 3; `model_b`: 2.

Milestone 1. Solve the divisibility and counting problem.

Step	Sub-task	Executor	Depends on
1	Directly solve the full problem by deriving the divisibility rules, counting valid permutations, and computing the final answer.	`model_b`	$\emptyset$
2	Re-derive the divisibility rule for 22 and verify the counting result.	`model_b`	$\emptyset$
3	Count the number of valid digit assignments satisfying the divisibility constraints.	`model_a`	[4]
4	Summarize the intermediate results and return the final answer.	`model_a`	[3]
5	Check whether the final answer is reasonable.	`model_a`	*Missing*

Analysis

The plan contains a dependency cycle between Steps 3 and 4, while Step 5 omits the required dependency field. These structural defects prevent the plan from defining a valid executable DAG. Steps 1 and 2 both use `model_b` for broad, overlapping reasoning, rather than assigning smaller verifiable sub-tasks. No final answer is included in the supplied trace.

Case B.3: E^3-ORCH Case on GAIA

Task. The Metropolitan Museum of Art has a portrait in its collection with an accession number of 29.100.5. Of the consecrators and co-consecrators of this portrait's subject as a bishop, what is the name of the one who never became pope?

Trace. 3 milestones; 9 planned steps. **Assignments.** `model_a`: 7; `model_b`: 2.

Milestone 1. Identify the portrait subject associated with accession number 29.100.5**.**

Step	Sub-task	Executor	Depends on
1	Search the Metropolitan Museum of Art collection using accession number `29.100.5`.	`model_a`	$\emptyset$
2	Extract the artwork title and portrait subject from the collection page.	`model_a`	[1]

Outcome. The portrait subject is Cardinal Fernando Niño de Guevara.

Milestone 2. Retrieve the consecrators and co-consecrators of Fernando Niño de Guevara.

Step	Sub-task	Executor	Depends on
1	Search for the episcopal consecration record of Fernando Niño de Guevara.	`model_a`	$\emptyset$
2	Extract the principal consecrator and co-consecrators from the retrieved record.	`model_a`	[1]
3	Verify that the extracted consecration record corresponds to the portrait subject to avoid entity mismatch.	`model_b`	[2]

Outcome. Fernando Niño de Guevara was consecrated by Pope Clement VIII, with Camillo Borghese and Alfonso Visconti serving as co-consecrators.

Milestone 3. Determine which consecrator or co-consecrator never became pope.

Step	Sub-task	Executor	Depends on
1	Verify that Clement VIII was pope.	`model_a`	$\emptyset$
2	Verify that Camillo Borghese later became Pope Paul V.	`model_a`	$\emptyset$
3	Identify the remaining consecrator or co-consecrator and verify whether he ever became pope.	`model_b`	[1, 2]
4	Return the final answer.	`model_a`	[3]

Final answer. Alfonso Visconti

Analysis

The trace uses `model_a` for collection lookup, extraction, and straightforward status checks. It invokes `model_b` twice: to check the identity of the consecration record and to verify the remaining candidate. The first two checks in Milestone 3 have no dependencies and can run independently. The recorded answer matches the baseline trace, with fewer steps and fewer assignments to `model_b`.

Case B.4: Baseline Case on GAIA

Task. The Metropolitan Museum of Art has a portrait in its collection with an accession number of 29.100.5. Of the consecrators and co-consecrators of this portrait's subject as a bishop, what is the name of the one who never became pope?

Trace. 3 milestones; 11 planned steps. **Assignments.** `model_a`: 4; `model_b`: 7.

Milestone 1. Identify the portrait subject and collect background information.

Step	Sub-task	Executor	Depends on
1	Search the Metropolitan Museum of Art collection using accession number `29.100.5` and identify the artwork title, artist, date, and portrait subject.	`model_b`	$\emptyset$
2	Search additional sources to reconfirm the same accession number, artwork title, and portrait subject.	`model_b`	[1]
3	Extract the final portrait subject from the verified collection information.	`model_a`	[2]

Outcome. The portrait subject is Cardinal Fernando Niño de Guevara.

Milestone 2. Retrieve the episcopal consecration information for Fernando Niño de Guevara.

Step	Sub-task	Executor	Depends on
1	Search for the episcopal consecration record of Fernando Niño de Guevara and extract the principal consecrator and co-consecrators.	`model_b`	$\emptyset$
2	Independently search another source for the same consecration record and compare the extracted names.	`model_b`	[1]
3	Merge the consecration information into a single list of names.	`model_a`	[1, 2]

Outcome. Fernando Niño de Guevara was consecrated by Pope Clement VIII, with Camillo Borghese and Alfonso Visconti serving as co-consecrators.

Milestone 3. Determine which consecrator or co-consecrator never became pope.

Step	Sub-task	Executor	Depends on
1	Verify the papal status of Pope Clement VIII.	`model_b`	$\emptyset$
2	Verify whether Camillo Borghese later became pope.	`model_b`	$\emptyset$
3	Verify whether Alfonso Visconti ever became pope.	`model_b`	$\emptyset$
4	Identify the consecrator or co-consecrator who never became pope from the verified status results.	`model_a`	[1, 2, 3]
5	Summarize the evidence and return the final answer.	`model_a`	[4]

Outcome. Clement VIII was pope, Camillo Borghese later became Pope Paul V, and Alfonso Visconti did not become pope.
Final answer. Alfonso Visconti

Analysis

The trace assigns seven of its eleven steps to `model_b`, including collection lookup, repeated retrieval, and all three papal-status checks. The supplied annotations identify redundant confirmation in Milestone 1, duplicate retrieval in Milestone 2, and an unnecessary final summarization step. The extra dependency in Milestone 1 also delays extraction. The trace reaches the same recorded answer as the corresponding E^3-ORCH case.

C LIMITATIONS AND FUTURE WORK

Our prompts, executor profiles, and evaluation benchmarks are predominantly English. The current results therefore do not establish whether the learned orchestration policy transfers to multilingual

tasks, code-switched inputs, or language-specific tools and executors. Multilingual settings may require the orchestrator to account explicitly for language proficiency and cross-lingual information integration.

The current evaluation further considers a limited number of executor configurations and assumes that the supplied capability and cost profiles are accurate and stable. In practice, executor behavior may vary across domains, prompts, and deployment conditions, while larger agent pools may substantially increase the orchestration search space. Evaluating robustness to noisy or evolving profiles and scaling to larger, more heterogeneous executor pools remain important directions for future work.

Finally, tree-structured rollouts increase training-time computation. The number of candidate trajectories grows with the number of initial trajectories (B), branches per expansion (N), and expansion rounds (E). Selective branching, early pruning, and trajectory reuse may reduce this training overhead.