DECOUPLED MULTI-AGENT ORCHESTRATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Learned orchestration can automatically construct effective language-model multi-agent systems, but existing approaches couple planning to fixed worker pools and train decomposition and collaboration from the same terminal outcome, limiting transfer and obscuring credit assignment. We introduce DEORCH, which separates worker-agnostic planning from concrete worker selection. Its two-stage planner first decomposes the task without worker information, then chooses collaboration operations using compact, worker-identity-free matchability feedback from the pool, enabling conditional credit assignment to decomposition and collaboration decisions. A lightweight matcher estimates worker suitability from behavior on a fixed probe set and adapts online with a contextual bandit, allowing new workers to be incorporated without retraining the planner or matcher. Across diverse in- and out-of-distribution tasks, DEORCH outperforms prior automatic MAS orchestration methods with fewer worker calls than competing learned orchestrators, remains effective when transferred to an entirely unseen worker pool without retraining, and shows consistent gains from both components.

1 INTRODUCTION

A multi-agent system (MAS) tackles a task with several language-model agents rather than one alone, letting them divide the work, verify one another, and combine their complementary strengths. Studies have shown that on some complex tasks a MAS outperforms each of its constituent agents (Wang et al., 2025; Kim et al., 2025; Nielsen et al., 2026). An expert-designed collaboration structure, specifying how the task is divided and how the agents work on it, often performs well; hand-designing one for every task, however, is impractical. Recent work therefore learns to build it automatically (Zhang et al., 2024; Dang et al., 2025; Gao et al., 2025; Cui et al., 2026).

A recent line of work trains an orchestrator that, given a task, decides how to decompose it into subtasks and how each subtask is handled by worker agents in a fixed pool (Gao et al., 2025; Chi et al., 2025; Cui et al., 2026). For instance, Conductor (Nielsen et al., 2026) learns, over a pool of powerful frozen workers, to decompose a task, assign each subtask to a worker, and specify how intermediate outputs are shared, training the orchestrator end-to-end on final task correctness. Relative to Conductor, MAS-Orchestra (Ke et al., 2026) does not model worker selection; it instead uses a fixed underlying worker and learns only to decompose the task and choose an operation for each subtask (e.g., a single worker, an ensemble, or a debate). Trained on outcome reward alone, these orchestrators learn effective coordination: by directing a fixed set of strong workers, a small orchestrator makes them collectively surpass any individual worker, and even rival much larger models, at a fraction of the cost.

Despite these successes, current automatic-MAS methods suffer from two limitations that reduce their effectiveness and practicality in realistic settings. **(1) The trained orchestrator struggles to exploit unseen workers.** Existing methods either ignore concrete worker selection or learn it directly into the orchestrator's policy (Ke et al., 2026; Nielsen et al., 2026; Gao et al., 2025). The former cannot exploit differences in workers' capabilities, while the latter cannot generalize its learned worker-selection strategy to unseen workers. This limitation is increasingly relevant in practice as new and stronger agents continually emerge: as the worker pool evolves, the orchestrator is either blind to which workers are best suited to each subtask or must be adapted to exploit newly available workers. **(2) A single outcome reward confounds credit assignment between decomposition and operation decisions.** The orchestrator makes two coupled decisions: how to decompose the task and which operation to run on each subtask (Ke et al., 2026; Cui et al., 2026). A failure can stem

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.

from either one: the same decomposition may succeed when a subtask is handled by an ensemble but fail when that subtask is left to a single worker, yet a single outcome reward cannot isolate which decision is responsible, leaving both with entangled learning signals. Moreover, once the subtasks are determined, the appropriate operation for each one depends on how well the available workers match that specific subtask: several moderately matched workers may favor an ensemble, whereas one clearly dominant match may favor a single worker. Without such subtask-specific information about the worker pool, the orchestrator cannot reliably distinguish these cases.

In this paper, we introduce DEORCH, an automatic MAS orchestration framework that addresses both limitations by decoupling planning from concrete worker selection. Specifically, DEORCH consists of two parts. **(1) A worker-agnostic, two-stage planner with matchability feedback.** The planner makes two decisions: how to decompose the task into subtasks and which collaboration operation to run on each. Rather than making both decisions at once, it proceeds in two stages. It first decomposes the task without observing the workers; the matcher then summarizes, for each subtask, how well the worker pool can support it through a compact, worker-identity-free *matchability* profile. Conditioned on this feedback, the planner chooses a collaboration operation for each subtask while keeping the decomposition fixed. This factorization enables separate credit assignment to decomposition and operation decisions during training, rather than training both from the same confounded outcome signal. It also localizes worker dependence: task decomposition is performed without worker information, while collaboration decisions adapt to the current pool through matchability feedback. Concrete worker selection remains entirely in the matcher, allowing the same planner to be applied to changed worker pools without retraining. **(2) A lightweight matcher combining probe-based profiling with online adaptation.** For each *(subtask, operation)* pair, the matcher selects suitable workers using both probe-based prior estimates of their capabilities and execution feedback collected during deployment. It first evaluates each worker on a fixed probe set, producing a behavioral profile from which its suitability for a new subtask is estimated based on semantically related probes. This provides a transferable cold-start estimate for new workers. A contextual bandit then refines these estimates from observed execution outcomes during deployment. Because worker-specific knowledge is confined to the matcher, a new or upgraded worker can be incorporated simply by evaluating it on the probe set, without retraining the planner or matcher.

We evaluate DEORCH across diverse in- and out-of-distribution tasks spanning math, code, multi-hop QA, tool use, and planning. DEORCH consistently outperforms existing automatic MAS orchestration baselines while using fewer worker calls, and remains effective when transferred to an entirely unseen worker pool without retraining. Ablations further validate both components: two-stage planning, matchability feedback, and conditional credit assignment improve orchestration, while probe-based matching, complementarity-aware worker selection, and online adaptation improve worker selection.

2 RELATED WORK

Automatic MAS Orchestration Automatic MAS design replaces hand-crafted collaboration with systems that automate decisions such as task decomposition, agent roles and prompts, and the flow of information among agents (Wang et al., 2026b; Li et al., 2026d;c; 2025; Shang et al., 2025). One line uses off-the-shelf LLMs to design or search over MAS structures, without training a dedicated orchestration model (Zhang et al., 2025b; Fourney et al., 2024; Li & Ramakrishnan, 2026; Lin et al., 2026; Liu et al., 2026; Yun et al., 2026; Zhang et al., 2025c; Wang et al., 2026d; Niu et al., 2025; Ke et al., 2025). For example, METAAGENT (Zhang et al., 2025c) asks an LLM to instantiate a set of agents and construct a finite-state machine that routes among them, while MASS (Zhou et al., 2026) optimizes both collaboration topologies and the prompts within them through a three-stage search pipeline. A second line uses non-LM controllers or optimization mechanisms to choose the MAS structure (Jiang et al., 2026; Li et al., 2026a; Xu et al., 2026; Li et al., 2026c;b; Wu et al., 2026; Wang et al., 2026a; Leong et al., 2025). For example, G-DESIGNER (Zhang et al., 2024) uses a GNN-based generator to decide which agents should communicate for a given task, while MAAS (Zhang et al., 2025a) learns a supernet containing many candidate agentic operators and activates only the parts needed for each query. These methods primarily optimize structural decisions rather than generating free-form task decompositions and subtask instructions in language. A third and more closely related line trains a language model specifically for orchestration (Dang et al., 2025; Wang et al., 2026c; Cui et al., 2026; Chi et al., 2025; Ruan et al., 2026; Huang et al., 2026). For

example, CONDUCTOR (Nielsen et al., 2026) uses reinforcement learning to train an LM that jointly writes the subtasks and assigns each subtask to a worker. MAS-ORCHESTRA (Ke et al., 2026) emits the entire orchestration in one decision step, instantiating for each subtask one of a fixed set of sub-agent types such as self-consistency, debate, or self-refine. FLOWREASONER (Gao et al., 2025) generates for each query a Python program defining the flow of a multi-agent system, with its subtasks handled by the same worker model under different roles. Unlike these approaches, which either omit concrete worker selection or couple it to planning, DEORCH keeps the planner worker-agnostic and delegates worker selection to a separate matcher.

LLM Routing and Model Selection LLM routing selects an appropriate model for each query from a candidate pool (Lu et al., 2024; Tsiourvas et al., 2025; Mei et al., 2025). Many existing routers are trained offline. One line trains a parametric router to predict which model best matches the input query (Ong et al., 2025; Chen et al., 2024; Song et al., 2025; Somerstep et al., 2025). For example, P2L (Frick et al., 2025) trains a language-model router whose output scores correspond to the candidate models, tying the router itself to the model pool. Another line avoids training such a router and instead estimates a model's performance on a new query from its behavior on a reference set (Shadid et al., 2025; Jitkrittum et al., 2026; Li, 2025b). AVENGERS-PRO (Zhang et al., 2025d), for example, maps a new query to semantically similar query clusters and routes according to the models' observed performance on them; a newly added model only needs to be evaluated on the reference set. A separate line studies online routing, treating model selection as a contextual-bandit problem and updating the router from deployment feedback, for example with LinUCB or Thompson sampling (Nguyen et al., 2025; Li, 2025a).

Credit Assignment in RL for LLMs Credit assignment arises whenever a single outcome reflects multiple decisions (Pignatelli et al., 2023). In LLM reasoning, an incorrect final answer gives little indication of which intermediate reasoning steps were useful or harmful, motivating process reward models (Lightman et al., 2024). In cooperative multi-agent learning, a shared team reward similarly obscures the contribution of individual agents, motivating value decomposition and counterfactual agent-level credit (Liang et al., 2024; Zhao & Xie, 2025). The same ambiguity appears in learned MAS orchestration, where a terminal outcome can entangle planning with worker selection and, within the planner, can further entangle task decomposition with operation decisions; recent work such as LEMON (Chen et al., 2026) localizes credit by counterfactually editing individual orchestration fields. To our knowledge, DEORCH is the first orchestration framework to use two-stage planning for RL credit assignment, separating decomposition from collaboration so that they receive distinct learning signals.

3 METHOD

Figure 1 gives an overview of DEORCH, which separates MAS orchestration into a worker-agnostic task planner and a separate matcher responsible for worker selection. Given a task, the language-model planner first decomposes it into a DAG of subtasks, whose edges encode dependencies. The matcher then summarizes how well the current worker pool can support each subtask through worker-identity-free matchability features, which the planner uses in a second pass to assign collaboration operations while keeping the decomposition fixed. Finally, the matcher binds each planned operation to suitable workers using its probe-based prior and online adaptation, and the resulting MAS is executed; execution outcomes are used to train the planner and update the matcher.

3.1 PROBLEM FORMULATION

Given a query x and a pool of worker agents $\mathcal{W} = \{w_1, \ldots, w_M\}$, automatic MAS orchestration constructs a multi-agent workflow for solving x. We represent an orchestration by

$$\mathcal{A} = (D = (S, E), \{(o_s, n_s)\}_{s \in S}, \{B_s\}_{s \in S}),$$

where D is a task-decomposition DAG whose nodes S are subtasks decomposed from x and whose directed edges E specify their dependencies. For each subtask s, $o_s \in \mathcal{O}$ denotes a collaboration operation from $\{\text{SINGLE}, \text{ENSEMBLE}, \text{DEBATE}, \text{SELECT}, \text{REVIEW}\}$, and n_s its required number of workers, while $B_s \subseteq \mathcal{W}$ denotes the concrete worker team assigned to execute it, with $|B_s| = n_s$. Executing $\mathcal{A}$ produces a final output y, whose quality is measured by an evaluator $Q(x, y)$. The

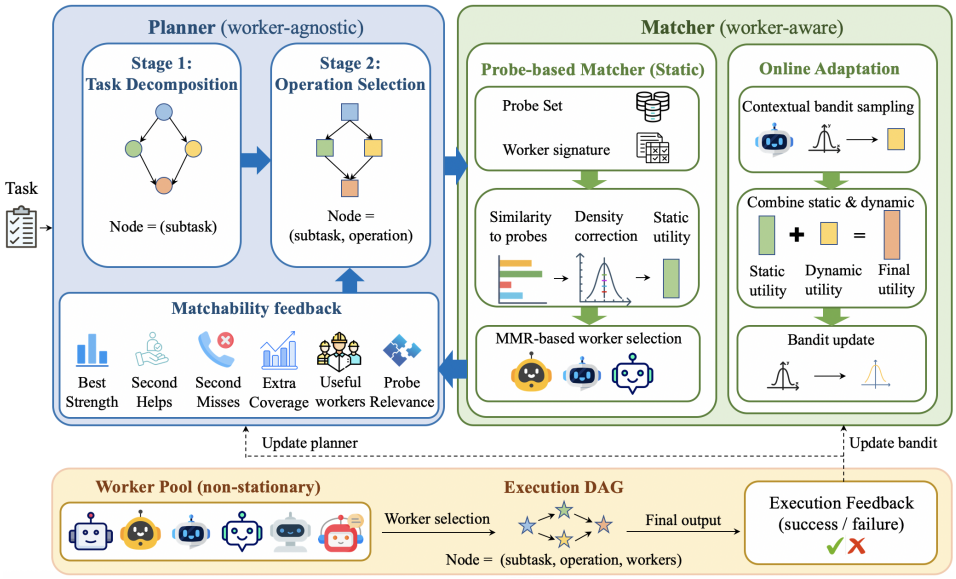


Figure 1: Overview of DEORCH.

orchestration problem therefore involves deciding how to decompose the task, how each subtask should be collaboratively executed, and which available workers should carry out that execution.

3.2 WORKER-AGNOSTIC TWO-STAGE PLANNER

Existing LM orchestrators often couple task planning with worker selection in the same policy (Nielsen et al., 2026; Cui et al., 2026), tying the orchestrator to a particular worker pool and requiring retraining when the pool changes. Our planner is instead worker-agnostic: it decides how to decompose the task and how each subtask should be collaboratively executed, while all concrete worker selection is delegated to the matcher. Workers can therefore be added to or removed from the pool without retraining the planner. The planner operates in two stages, separating task decomposition from collaboration planning for two reasons. First, a shared terminal reward entangles the two decisions: failure does not reveal whether the decomposition or collaboration strategy was responsible. Separating the two decisions allows collaboration choices to be sampled and compared while holding the decomposition fixed, yielding more targeted credit for collaboration decisions during RL training. Second, collaboration should depend on how well the current worker pool supports each subtask. For example, a subtask that is easily handled by the available workers may require only a single worker, whereas a poorly matched subtask may benefit from ensemble execution. The second stage therefore conditions operation selection on worker-identity-free matchability feedback, adapting collaboration to the pool without exposing worker identities.

Stage I: task decomposition. Given an input query x, the planner generates a task-decomposition DAG $D = (S, E) \sim \pi_\theta^{\text{dec}}(\cdot \mid x)$, where $S = \{s_1, \ldots, s_L\}$ contains natural-language subtasks and E specifies their dependencies. At this stage, the planner decomposes the task from the query alone, determining what should be solved and in what dependency order, while leaving collaboration planning and worker selection to subsequent stages.

Stage II: collaboration planning. After generating the decomposition, the matcher evaluates how well the worker pool supports each subtask and returns a matchability feature $m_s = m(s; \mathcal{W})$, a fixed-dimensional, worker-identity-free summary of pool support (Section 3.3). Conditioned on the query, fixed decomposition, and matchability features, the planner performs a second generation:

$$\{(o_s, n_s)\}_{s \in S} \sim \pi_\theta^{\text{op}}\left(\cdot \mid x, D, \{m_s\}_{s \in S}\right),$$

where $o_s \in \mathcal{O}$ specifies the collaboration operation used for subtask s and n_s denotes the required team size. Stage II keeps the decomposition fixed and augments each subtask with (o_s, n_s), adapt-

ing collaboration to the current pool through matchability feedback while leaving concrete worker selection to the matcher.

Collaboration operations. We consider a compact operation set $\mathcal{O} = \{\text{SINGLE}, \text{ENSEMBLE}, \text{DEBATE}, \text{SELECT}, \text{REVIEW}\}$. Each operation determines how the assigned worker or worker team interacts when solving s. Specifically, SINGLE uses one worker; ENSEMBLE aggregates responses from multiple workers; DEBATE lets multiple workers exchange and refine their responses; SELECT uses an adjudicator to choose among candidate responses; and REVIEW uses reviewers to critique and refine an initial response. By composing these operations over the subtask DAG, the planner can instantiate common MAS structures.

3.3 PROBE-BASED MATCHER WITH ONLINE ADAPTATION

The matcher combines a static probe-based matcher with an online contextual bandit. The static matcher estimates worker suitability for a subtask from performance on semantically related probes, while the online bandit refines these estimates from deployment feedback. The static matcher is used for planner training and deployment, while online adaptation is used only at deployment.

Probe-based matching with density correction. We construct a fixed probe set of N diverse tasks. Each worker w is evaluated once on the same probes, producing a behavioral signature $\mathbf{o}_w = (o_{w,1}, \ldots, o_{w,N}) \in \{0,1\}^N$, where $o_{w,j} = 1$ if worker w succeeds on probe j and 0 otherwise. Given a subtask s, we estimate worker suitability from its outcomes on semantically related probes. Let e_s and e_j denote the embeddings of s and probe j. We define the subtask–probe relevance as $\kappa_j(s) = \exp((\cos(e_s, e_j) - 1)/h_{\text{rel}})$, where smaller h_{rel} focuses on more similar probes. Directly normalizing these weights can overemphasize semantic regions containing many near-duplicate probes. We therefore estimate the local probe density

$$q_j = \sum_{\ell=1}^{N} \exp\left(\frac{\cos(e_j, e_\ell) - 1}{h_{\text{den}}}\right), \qquad \bar{q}_j = \frac{q_j}{\text{median}_\ell\, q_\ell},$$

where h_{den} controls the neighborhood scale. The final probe weights and worker suitability are

$$\alpha_j(s) = \frac{\kappa_j(s)(\bar{q}_j + \epsilon)^{-\nu}}{\sum_{\ell=1}^{N} \kappa_\ell(s)(\bar{q}_\ell + \epsilon)^{-\nu}}, \qquad \hat{u}_w(s) = \sum_{j=1}^{N} \alpha_j(s) o_{w,j}.$$

The density correction reduces the influence of locally redundant probes while preserving genuinely distinct coverage. Since $\alpha_j(s)$ depends only on the subtask and the fixed probe set, a new worker can be scored directly from its probe signature without retraining the matcher.

Matchability feedback. The second-stage planner should know how well the current worker pool can support a subtask without observing worker identities. We therefore summarize the pool with a fixed six-dimensional feature $m(s) = [B(s), U(s), L(s), G(s), D(s), C(s)]$. The six coordinates capture complementary aspects of pool capability: B measures the strength of the best-matched single worker; U and L describe the benefit and risk of adding a complementary second worker; G and D summarize the amount and breadth of complementary capability available across additional workers; and C measures how well the probe set supports the estimate for the current subtask. Importantly, these statistics describe the pool without encoding worker identities or its size. Hence the planner receives the same fixed interface as workers are added or removed, while the matcher recomputes the summary from the current pool.

Let $w_1 = \arg\max_w \hat{u}_w(s)$ be the best-matched worker, so that $B(s) = \hat{u}_{w_1}(s)$. Starting from w_1, we greedily add workers according to their marginal increase in weighted probe coverage, up to a maximum team size K. Let g_k denote the marginal coverage gain from the k-th addition. The first added worker, denoted w_2, maximally covers the probe failures of w_1, giving

$$U(s) = \sum_j \alpha_j(s)(1 - o_{w_1,j}) o_{w_2,j}, \qquad L(s) = \sum_j \alpha_j(s) o_{w_1,j}(1 - o_{w_2,j}),$$

where U measures rescued failure mass and L measures converse spoilage mass. In particular, $g_1 = U(s)$. The resulting gains $g_1, \ldots, g_{K-1}$ are summarized as

$$G(s) = \sum_k g_k, \qquad D(s) = \frac{G(s)^2}{\sum_k g_k^2}, \qquad C(s) = \sum_j \alpha_j(s) \cos(e_s, e_j).$$

Here, G measures the total complementary coverage available beyond the best single worker, D reflects how broadly that coverage is distributed across additional workers, and C measures the semantic support provided by the probes used for the estimate.

Operation-specific worker selection. Once the planner specifies (s, o_s, n_s), the matcher selects workers using subtask suitability and behavioral complementarity. For SINGLE, we select the worker with the highest suitability score $\hat{u}_w(s)$. For multi-worker operations, we use maximal marginal relevance (MMR) (Carbonell & Goldstein, 1998) for team construction. We initialize B with the highest-suitability worker and then greedily add workers until $|B| = n_s$. We define the directional complementarity of workers w and v and select each additional worker according to

$$\delta_{w \leftarrow v}(s) = \sum_{j=1}^{N} \alpha_j(s)(1 - o_{v,j}) o_{w,j}, \qquad w^\star = \arg\max_{w \notin B} \left[\lambda_o \hat{u}_w(s) - (1 - \lambda_o) \max_{v \in B} (1 - \delta_{w \leftarrow v}(s)) \right].$$

This criterion balances competence with failure-region complementarity to the selected team B. We use it for ENSEMBLE and DEBATE. Operations with asymmetric roles use specialized rules: SELECT first assigns the highest-suitability worker as the adjudicator and uses MMR to choose the remaining proposers, whereas REVIEW assigns the highest-suitability worker as the producer and greedily selects reviewers that best cover its failure region.

Online adaptation. The probe-based matcher provides a cold-start estimate from worker behavior, while deployment experience provides evidence of worker utility on the target distribution. We therefore augment the static matcher with contextual Thompson sampling. Each worker w has a probe-derived capability representation $g(w)$, and we maintain a Gaussian posterior over a shared parameter η modeling context-dependent task–worker interactions. Before each MAS execution, we sample $\tilde{\eta}$ from the posterior and combine the probe-based estimate with the interaction between the subtask representation $z(s)$ and worker representation $g(w)$ to form the deployment-adapted suitability $\tilde{u}_w(s) = \hat{u}_w(s) + \tilde{\eta}^\top (z(s) \odot g(w))$. At deployment, $\tilde{u}_w(s)$ replaces $\hat{u}_w(s)$ in the operation-specific worker-selection rules, while the probe-derived complementarity statistics remain unchanged. After the MAS is executed, its terminal utility is treated as noisy bandit feedback for the selected worker assignment and used to update the shared posterior. The bandit does not directly attribute a single outcome to individual workers; rather, repeated feedback across tasks and worker assignments allows deployment-specific task–worker interactions to be learned over time.

3.4 PLANNER TRAINING WITH CONDITIONAL CREDIT ASSIGNMENT

We post-train the planner from terminal execution outcomes using nested conditional rollouts. For each query, we sample multiple decompositions and, for each fixed decomposition, multiple collaboration plans. Comparing collaboration plans under the same decomposition provides credit for Stage II, while averaging their outcomes estimates how good that decomposition is for Stage I. This yields separate learning signals for the two planning stages from the same terminal outcomes.

Conditional credit assignment. For each training query x, we sample J decompositions and, for each fixed decomposition, k collaboration plans:

$$D_j \sim \pi_\theta^{\text{dec}}(\cdot \mid x), \qquad j = 1, \ldots, J,$$
$$O_{j,l} \sim \pi_\theta^{\text{op}}\big(\cdot \mid x, D_j, \{m_s\}_{s \in S_j}\big), \quad l = 1, \ldots, k.$$

Each resulting plan is assigned concrete workers by the matcher and executed, yielding terminal quality $Q_{j,l} = Q(x, y_{j,l})$. For Stage II, all k rollouts under the same D_j share the same upstream decomposition, so their relative outcomes provide credit for the collaboration decisions. We first form a cost-adjusted reward and then compute its relative advantage within the group:

$$R_{j,l}^{\text{op}} = Q_{j,l} \exp(-\beta c_{j,l}), \qquad A_{j,l}^{\text{op}} = R_{j,l}^{\text{op}} - \frac{1}{k} \sum_{l'=1}^{k} R_{j,l'}^{\text{op}},$$

where $c_{j,l}$ is the number of worker calls incurred by rollout (j, l) and β controls the cost penalty. For Stage I, we first average over downstream collaboration choices and then compare the resulting decomposition-level scores:

$$\bar{Q}_j = \frac{1}{k}\sum_{l=1}^{k} Q_{j,l}, \qquad A_j^{\mathrm{dec}} = \bar{Q}_j - \frac{1}{J}\sum_{j'=1}^{J} \bar{Q}_{j'}.$$

We use A^{op} and A^{dec} to optimize the corresponding Stage-II and Stage-I generations, respectively.

Cost-aware collaboration planning. Existing learned orchestrators such as CONDUCTOR and MAS-ORCHESTRA primarily optimize task quality without explicitly accounting for collaboration cost. Since more intensive collaboration typically incurs more worker calls and can also improve performance, a quality-only objective may systematically favor heavier operations. This cost–quality tradeoff is directly controlled by Stage II. In contrast, Stage I determines the task decomposition, and decomposition granularity by itself does not indicate solution quality: finer or coarser decompositions can each be preferable depending on the task. Our two-stage formulation therefore allows us to train Stage I with task quality alone while applying the cost-aware objective only to Stage II. The multiplicative reward further ensures that lower cost is valuable only when accompanied by task success: a failed rollout does not receive a higher reward merely because it uses fewer worker calls.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models. We initialize the planner from `Qwen3-8B` and perform full-parameter RL post-training. Unless otherwise specified, all methods operate over the same heterogeneous pool of six worker models: `Qwen3-4B-Instruct-2507`, `gemma-4-12B-it`, `NVIDIA-Nemotron-Nano-9B-v2`, `Qwen3.5-9B`, `granite-4.1-8b`, and `Ministral-3-14B-Instruct-2512`. We use `Qwen3-Embedding-8B` as the text encoder for semantic matching and retrieval.

Data and benchmarks. We use disjoint data for probe construction, planner post-training, and final evaluation. We evaluate in-distribution on held-out MuSiQue and TravelPlanner examples, and out-of-distribution on Omni-MATH-2, LiveCodeBench v6, BBEH, NaturalPlan, PlanBench-mystery, and GAIA, spanning math, code, multi-hop reasoning, tool use, and planning. Probe and training datasets, data sources, splits, and sample counts are provided in Appendix B.1.

Baselines. We compare against several complementary baselines. BEST SINGLE AGENT uses `gemma-4-12B-it`, the best-performing worker in hindsight. As a fixed hand-designed MAS baseline, we include INDEPENDENT coordination from prior work (Kim et al., 2025). FRONTIER-PLANNER uses Qwen3.8-27B as an off-the-shelf planner, with the rest of the system following DEORCH. UNTRAINED PLANNER uses the same planner initialization as DEORCH but without orchestration-specific RL post-training. We further compare with learned automatic orchestrators, including CONDUCTOR (Nielsen et al., 2026) and MAS-ORCHESTRA (Ke et al., 2026), adapted to our worker setting. Additional baseline details are provided in Appendix B.3.

4.2 MAIN RESULTS

Table 1 summarizes the main results. DEORCH achieves the best overall performance across both in-distribution and out-of-distribution benchmarks. It substantially outperforms BEST SINGLE AGENT and INDEPENDENT, showing that its gains cannot be explained by relying on the strongest worker or by simply executing all workers in parallel. Both FRONTIER-PLANNER and UNTRAINED PLANNER perform considerably worse, indicating that strong general-purpose planning or an untrained planner does not substitute for orchestration-specific post-training. Compared with learned automatic orchestrators, DEORCH consistently outperforms CONDUCTOR and MAS-ORCHESTRA, demonstrating the effectiveness and competitiveness of our approach. Moreover, DEORCH uses fewer worker calls than both learned orchestrators, indicating that its performance gains do not come from making more worker calls.

Table 1: Main results across in-distribution and out-of-distribution benchmarks.

	In-Distribution		Out-of-Distribution						Overall	
Method	MuSiQue	TP	Omni	LCB	BBEH	NP	PB-M	GAIA	Avg.	Calls ↓
BEST SINGLE AGENT	72.8	13.6	42.5	69.9	38.4	51.1	19.4	6.1	39.2	1.00
INDEPENDENT	72.0	9.2	47.4	76.4	45.2	63.1	25.9	28.5	46.0	7.00
FRONTIER-PLANNER	59.1	3.5	38.9	71.0	37.4	45.5	9.1	28.5	36.6	4.67
UNTRAINED PLANNER	60.2	3.3	38.1	70.4	31.8	50.4	25.2	26.7	38.3	4.92
CONDUCTOR	70.3	9.9	41.2	75.0	37.1	52.9	38.0	27.9	44.0	6.16
MAS-ORCHESTRA	73.4	15.1	49.0	76.9	45.0	61.8	41.6	29.7	49.1	5.63
DEORCH	76.9	17.8	51.7	79.2	47.9	64.7	43.1	30.9	51.5	5.35

4.3 GENERALIZATION TO AN UNSEEN WORKER POOL

Table 2 evaluates generalization to an entirely unseen four-worker pool (Appendix B.3), with no planner or matcher retraining and online adaptation disabled. DEORCH remains effective, outperforming both BEST SINGLE AGENT and INDEPENDENT by clear margins. In contrast, CONDUCTOR degrades from 44.0 to 40.9: its advantage over BEST SINGLE AGENT nearly disappears, its gap to INDEPENDENT widens, and its gap to DEORCH increases from 7.5 to 9.3 points. These results support the benefit of decoupling planning from concrete worker selection.

Table 2: Performance under an unseen worker pool.

	In-Distribution		Out-of-Distribution						Overall
Method	MuSiQue	TP	Omni	LCB	BBEH	NP	PB-M	GAIA	Avg.
BEST SINGLE AGENT	69.0	6.2	45.9	75.9	47.7	40.3	29.0	12.1	40.8
INDEPENDENT	72.4	5.0	50.6	81.2	48.4	60.4	41.6	20.0	47.5
FRONTIER-PLANNER	61.4	3.1	44.4	76.3	46.8	52.2	18.1	21.8	40.5
CONDUCTOR	63.7	4.6	45.5	75.5	46.2	48.8	23.7	18.8	40.9
DEORCH	75.5	11.3	52.3	83.0	49.7	62.3	44.3	23.0	50.2

4.4 ANALYSIS OF THE PLANNER

We construct controlled variants to isolate the contributions of the planner design and training objective. JOINT PLANNING generates decomposition and collaboration decisions in one pass under a shared terminal reward. TWO-STAGE separates the two decisions but uses neither matchability feedback nor conditional credit. + MATCHABILITY provides the Stage-II planner with $m(s)$, and + CONDITIONAL CREDIT further introduces our nested credit assignment. The full DEORCH additionally applies the cost-aware Stage-II utility. We also evaluate DECOMPOSITION + SINGLE, which preserves the learned decomposition but fixes every subtask to SINGLE, measuring how much performance remains without Stage-II collaboration choices. All trained variants use the same planner backbone, worker pool, and rollout budget. Table 3 shows consistent improvements as the planner components are introduced. Two-stage planning and matchability feedback each improve performance, while conditional credit assignment yields the largest gain, supporting separate credit for decomposition and collaboration decisions. DECOMPOSITION + SINGLE still achieves an overall average of 50.4, showing that the learned decomposition alone is already strong, while learned Stage-II collaboration provides additional gains. Finally, the cost-aware objective reduces worker calls from 7.78 to 5.35 with only a modest decrease in average performance from 51.9 to 51.5, substantially improving the quality–cost trade-off.

4.5 ANALYSIS OF THE MATCHER

Offline worker matching. We evaluate the static matcher independently of the learned planner. For in-distribution evaluation, examples from each probe source are randomly split 80/20 for probe construction and held-out evaluation. We additionally evaluate leave-one-source-out (LOSO) generalization on five representative sources by excluding the target source from the probe set. We

Table 3: Ablation of the planner.

Variant	ID Avg.	OOD Avg.	Avg.	Calls ↓
Joint Planning	43.2	50.2	48.5	6.29
Two-Stage	43.8	50.9	49.2	7.33
+ Matchability	44.3	51.4	49.6	7.52
+ Conditional Credit	47.7	53.4	51.9	7.78
Decomposition + Single	46.1	51.8	50.4	1.68
DEORCH	47.4	52.9	51.5	5.35

compare against random routing, the strongest single worker, majority voting over all six workers, probe matching without density correction, and ORACLE SINGLE, which selects a successful worker in hindsight as an upper bound. Table 4 shows that the full probe matcher achieves the strongest non-oracle performance in both settings. Its advantage narrows under LOSO-OOD evaluation, where no probe examples from the target source are available, but it still outperforms the strongest single-worker reference. Density correction provides consistent gains in both settings, supporting its role in reducing the influence of locally redundant probes. Finally, the sizable gap to ORACLE SINGLE indicates substantial remaining headroom in worker selection.

Table 4: Offline evaluation of worker matching.

Method	ID Avg. ↑	LOSO-OOD Avg. ↑	Calls ↓
Random	59.44	69.73	1
BEST SINGLE WORKER	68.33	75.26	1
Majority vote	67.92	74.59	6
Probe Matcher w/o Density	69.70	75.36	1
Probe Matcher	70.24	75.45	1
ORACLE SINGLE	82.68	87.17	1

Table 5: Online adaptation with increasing deployment feedback.

Adaptation Examples	0	3K	6K	9K	12K	15K
Overall Avg.	51.3	51.7	52.1	52.1	52.6	52.9

Online adaptation. We finally evaluate whether contextual bandit adaptation improves the matcher's worker-selection ability during deployment. We split the evaluation data into an adaptation stream and a disjoint held-out evaluation set, with the split detailed in Appendix B.2. The planner and the probe-based component of the matcher remain fixed throughout the experiment. Adaptation examples are processed sequentially, and the shared Thompson-sampling posterior is updated after each execution using the observed terminal outcome. After every 3,000 adaptation examples, we evaluate the current matcher on the fixed held-out set. Performance is computed separately on each benchmark and macro-averaged across the eight benchmarks. The zero-example checkpoint corresponds to the static probe matcher without deployment feedback. Table 5 shows that performance generally improves as deployment feedback accumulates, increasing from 51.3 without online adaptation to 52.9 after 15K adaptation examples. This result demonstrates the effectiveness of our contextual bandit in progressively improving worker selection during deployment.

5 CONCLUSION

We introduced DEORCH, a multi-agent orchestration framework that decouples worker-agnostic planning from concrete worker selection and uses two-stage planning for separate credit assignment to decomposition and collaboration decisions. Across diverse tasks, DEORCH outperforms fixed and learned MAS baselines and remains effective under worker-pool changes without retraining.

AI USE STATEMENT

Generative AI tools were only used to assist with language editing and polishing of the manuscript, including improving clarity, readability, grammar, and phrasing. All AI-assisted edits were reviewed by the authors. The authors take full responsibility for the final content of this work.

REFERENCES

Jaime G Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, 1998.

Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. Routerdc: Query-based router by dual contrastive learning for assembling large language models. *Advances in Neural Information Processing Systems*, 37:66305–66328, 2024.

Xudong Chen, Yixin Liu, Hua Wei, and Kaize Ding. Lemon: Learning executable multi-agent orchestration via counterfactual reinforcement learning. *arXiv preprint arXiv:2605.14483*, 2026.

Zewen Chi, Li Dong, Qingxiu Dong, Yaru Hao, Xun Wu, Shaohan Huang, and Furu Wei. The era of agentic organization: Learning to organize with language models. *arXiv preprint arXiv:2510.26658*, 2025.

Zhiqing Cui, Haotong Xie, Jiahao Yuan, Cheng Yang, Hanqing Wang, Yuxin Wu, Yifan Wu, Siru Zhong, Tao Yu, Yifu Guo, et al. Uno-orchestra: Parsimonious agent routing via selective delegation. *arXiv preprint arXiv:2605.05007*, 2026.

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 *Advances in Neural Information Processing Systems*, volume 38, 2025.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

Evan Frick, Connor Chen, Joseph Tennyson, Tianle Li, Wei-Lin Chiang, Anastasios N Angelopoulos, and Ion Stoica. Prompt-to-leaderboard. *arXiv preprint arXiv:2502.14855*, 2025.

Hongcheng Gao, Yue Liu, Yufei He, Longxu Dou, Chao Du, Zhijie Deng, Bryan Hooi, Min Lin, and Tianyu Pang. Flowreasoner: Reinforcing query-level meta-agents. *arXiv preprint arXiv:2504.15257*, 2025.

Jing Huang, Lidong Zhang, Mutian Bao, Yadong Li, Xingzhong Xu, Jinjian Zhang, Jie Liu, Ming Kong, and Qiang Zhu. Mas-architect: Declarative multi-agent system design via separation of concerns. In *Forty-third International Conference on Machine Learning*, 2026.

Eric Hanchen Jiang, Levina Li, Rui Sun, Xiao Liang, Yubei Li, Yuchen Wu, Haozheng Luo, Hengli Li, Zhi Zhang, Zhaolu Kang, et al. Agent q-mix: Selecting the right action for llm multi-agent systems through reinforcement learning. *arXiv preprint arXiv:2604.00344*, 2026.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, et al. Universal model routing for efficient llm inference. In *International Conference on Learning Representations*, volume 2026, pp. 10169–10218, 2026.

Zixuan Ke, Austin Xu, Yifei Ming, Xuan-Phi Nguyen, Ryan Chin, Caiming Xiong, and Shafiq Joty. Mas-zero: Designing multi-agent systems with zero supervision. *arXiv preprint arXiv:2505.14996*, 2025.

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. *arXiv preprint arXiv:2601.14652*, 2026.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, et al. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

Hui Yi Leong, Yuheng Li, Yuqing Wu, Wenwen Ouyang, Wei Zhu, Jiechao Gao, and Wei Han. Amas: Adaptively determining communication topology for llm-based multi-agent system. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 2061–2070, 2025.

Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. Agent-oriented planning in multi-agent systems. In *International Conference on Learning Representations*, volume 2025, pp. 19495–19517, 2025.

Haoran Li, Shulun Chen, Shaoyuan Sun, and Hanchen Wang. Multi-agent coordination adaptation via structure-guided orchestration. *arXiv preprint arXiv:2605.25746*, 2026a.

Sha Li and Naren Ramakrishnan. Experience as a compass: Multi-agent rag with evolving orchestration and agent prompts. *arXiv preprint arXiv:2604.00901*, 2026.

Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 23142–23150, 2026b.

Shiyuan Li, Yixin Liu, Yu Zheng, Mei Li, Quoc Viet Hung Nguyen, and Shirui Pan. Ofa-mas: One-for-all multi-agent system topology design based on mixture-of-experts graph generative models. In *Proceedings of the ACM Web Conference 2026*, pp. 1333–1344, 2026c.

Yang Li. Llm bandit: Cost-efficient llm generation via preference-conditioned dynamic routing. *arXiv preprint arXiv:2502.02743*, 2025a.

Yang Li. Rethinking predictive modeling for llm routing: When simple knn beats complex learned routers. *arXiv preprint arXiv:2505.12601*, 2025b.

Yu Li, Lehui Li, Zhihao Wu, Qingmin Liao, Jianye Hao, Kun Shao, and Fengli Xu. Agentswift: Efficient llm agent design via value-guided hierarchical search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 31843–31851, 2026d.

Yongheng Liang, Hejun Wu, Haitao Wang, and Hao Cai. Asynchronous credit assignment for multi-agent reinforcement learning. *arXiv preprint arXiv:2408.03692*, 2024.

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

Hehai Lin, Yu Yan, Zixuan Wang, Bo Xu, Sudong Wang, Weiquan Huang, Ruochen Zhao, Minzhi Li, and Chengwei Qin. Unified-mas: Universally generating domain-specific nodes for empowering automatic multi-agent systems. *arXiv preprint arXiv:2603.21475*, 2026.

Guangyi Liu, Haojun Lin, Huan Zeng, Heng Wang, and Quanming Yao. Mas-on-the-fly: Dynamic adaptation of llm-based multi-agent systems at test time. *arXiv preprint arXiv:2602.13671*, 2026.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. Routing to the expert: Efficient reward-guided ensemble of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1964–1974, 2024.

Kai Mei, Wujiang Xu, Minghao Guo, Shuhang Lin, and Yongfeng Zhang. Omnirouter: Budget and performance controllable multi-llm routing. *ACM SIGKDD Explorations Newsletter*, 27(2): 107–116, 2025.

Duy M. H. Nguyen, Archiki Prasad, Elias Stengel-Eskin, and Mohit Bansal. LASeR: Learning to adaptively select reward models with multi-arm bandits. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3895.

Stefan Nielsen, Edoardo Cetin, Peter Schwendeman, Qi Sun, Jinglue Xu, and Yujin Tang. Learning to orchestrate agents in natural language with the conductor. In *International Conference on Learning Representations*, volume 2026, pp. 135686–135724, 2026.

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: Modularized agentic workflow automation. In *International Conference on Learning Representations*, volume 2025, pp. 74949–74977, 2025.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

Eduardo Pignatelli, Johan Ferret, Matthieu Geist, Thomas Mesnard, Hado van Hasselt, Olivier Pietquin, and Laura Toni. A survey of temporal credit assignment in deep reinforcement learning. *arXiv preprint arXiv:2312.01072*, 2023.

Jianhao Ruan, Zhihao Xu, Yiran Peng, Fashen Ren, Zhaoyang Yu, Xinbing Liang, Jinyu Xiang, Yongru Chen, Bang Liu, Chenglin Wu, et al. Aorchestra: Automating sub-agent creation for agentic orchestration. *arXiv preprint arXiv:2602.03786*, 2026.

Ahmad Shadid, Rahul Kumar, and Mohit Mayank. Ori: O routing intelligence. *arXiv preprint arXiv:2502.10051*, 2025.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. In *International Conference on Learning Representations*, volume 2025, pp. 3841–3865, 2025.

Seamus Somerstep, Felipe Maia Polo, Allysson Flavio Melo de Oliveira, Prattyush Mangal, Mírian Silva, Onkar Bhardwaj, Mikhail Yurochkin, and Subha Maity. Carrot: A cost aware rate optimal router. *arXiv preprint arXiv:2502.03261*, 2025.

Wei Song, Zhenya Huang, Cheng Cheng, Weibo Gao, Bihan Xu, GuanHao Zhao, Fei Wang, and Runze Wu. Irt-router: Effective and interpretable multi-llm routing via item response theory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15629–15644, 2025.

Asterios Tsiourvas, Wei Sun, and Georgia Perakis. Causal LLM routing: End-to-end regret minimization from observational data. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Jiye Wang, Yu Wang, Jianbin Li, Shiduo Yang, Kenan Guo, and Yuanhe Zhao. Neuralfsm: Adaptive multi-agent coordination via learning finite-state execution policy. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33414–33436, 2026a.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Y Zou. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations*, volume 2025, pp. 33944–33963, 2025.

Kun Wang, Guibin Zhang, ManKit Ye, Xinyu Deng, Dongxia Wang, Xiaobin Hu, Jinyang Guo, Yang Liu, and Yufei Guo. MAS^2: Self-generative, self-configuring, self-rectifying multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pp. 113586–113613, 2026b.

Siyu Wang, Ruotian Lu, Zhihao Yang, Yuchao Wang, Yanzhou Zhang, Lei Xu, Qimin Xu, Guojun Yin, Cailian Chen, and Xinping Guan. Agentconductor: Topology evolution for multi-agent competition-level code generation. *arXiv preprint arXiv:2602.17100*, 2026c.

Yimeng Wang, Jiaxing Zhao, Hongbin Xie, Hexing Ma, Yuzhen Lei, Shuangxue Liu, Xuan Song, Zichen Zhang, and Haoran Zhang. Metagen: Self-evolving roles and topologies for multi-agent llm reasoning. *arXiv preprint arXiv:2601.19290*, 2026d.

Tongtong Wu, Yanming Li, Ziye Tang, Chen Jiang, Linhao Luo, Guilin Qi, Shirui Pan, and Gholamreza Haffari. Card: Towards conditional design of multi-agent topological structures. *arXiv preprint arXiv:2603.01089*, 2026.

Chengdong Xu, Kaiqiang Ke, Ziheng Liu, Jiaqi Wei, Zibo Shao, Weile Guo, and Chao Yu. Evomas: Learning execution-time workflows for multi-agent systems. *arXiv preprint arXiv:2605.08769*, 2026.

Sukwon Yun, Jie Peng, Pingzhi Li, Wendong Fan, Jie Chen, James Y Zou, Guohao Li, and Tianlong Chen. Graph-of-agents: A graph-based framework for multi-agent llm collaboration. In *International Conference on Learning Representations*, volume 2026, pp. 19745–19760, 2026.

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. *arXiv preprint arXiv:2410.11782*, 2024.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025a.

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

Yaolun Zhang, Xiaogeng Liu, and Chaowei Xiao. Metaagent: Automatically constructing multi-agent systems based on finite state machines. *arXiv preprint arXiv:2507.22606*, 2025c.

Yiqun Zhang, Hao Li, Jianhao Chen, Hangfan Zhang, Peng Ye, Lei Bai, and Shuyue Hu. Beyond gpt-5: Making llms cheaper and better via performance-efficiency optimized routing. In *Proceedings of the 2025 7th International Conference on Distributed Artificial Intelligence*, pp. 122–129, 2025d.

Xutong Zhao and Yaqi Xie. Multi-level advantage credit assignment for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2508.06836*, 2025.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Arik. Multi-agent design: Optimizing agents with better prompts and topologies. In *International Conference on Learning Representations*, volume 2026, pp. 15844–15872, 2026.

A Matcher Details

A.1 Online Matcher Adaptation

We implement deployment-time adaptation using linear contextual Thompson sampling with aggregate execution feedback. Online adaptation changes only concrete worker selection; the planner-facing matchability features remain probe-derived.

Let $z(\cdot) \in \mathbb{R}^d$ denote the frozen embedding produced by the text encoder for either a subtask or a probe. Using the probe-density statistic $\bar{q}_j$ defined in Section 3.3, we define a density-correction weight

$$\rho_j = (\bar{q}_j + \epsilon)^{-\nu}, \qquad Z = \sum_{j=1}^{N} \rho_j.$$

For each worker w, its probe-derived capability representation is

$$g(w) = \frac{1}{Z} \sum_{j=1}^{N} \rho_j \, o_{w,j} \, z(p_j),$$

where p_j denotes probe j. The density correction reduces the influence of semantically redundant probe regions, while the worker-independent normalization Z preserves differences in overall probe performance across workers.

For a subtask–worker pair, we define the dimension-aligned interaction

$$\phi(s, w) = z(s) \odot g(w).$$

Rather than maintaining a separate online parameter for every worker, we use a single shared parameter $\eta \in \mathbb{R}^d$. Given a posterior sample $\tilde{\eta}$, the deployment-adapted suitability is

$$\tilde{u}_w(s) = \hat{u}_w(s) + \tilde{\eta}^\top \phi(s, w).$$

The adapted score replaces $\hat{u}_w(s)$ in the operation-specific worker-selection rules, while probe-derived complementarity statistics remain unchanged.

For execution t, let $B_{t,s}$ denote the workers assigned to subtask s and

$$A_t = \sum_s |B_{t,s}|.$$

We aggregate the task–worker interactions and static suitability estimates of the selected assignment as

$$x_t = \frac{1}{A_t} \sum_s \sum_{w \in B_{t,s}} \phi(s, w), \qquad \bar{u}_t = \frac{1}{A_t} \sum_s \sum_{w \in B_{t,s}} \hat{u}_w(s).$$

We model the terminal utility as

$$r_t = \bar{u}_t + x_t^\top \eta + \epsilon_t, \qquad \epsilon_t \sim \mathcal{N}(0, \sigma^2).$$

We place a zero-mean Gaussian prior

$$\eta \sim \mathcal{N}(0, \Lambda_0^{-1}).$$

Treating $\bar{u}_t$ as a known probe-based offset, we define

$$y_t = r_t - \bar{u}_t$$

and perform the standard Bayesian linear-regression update

$$\Lambda_t = \Lambda_{t-1} + \sigma^{-2} x_t x_t^\top, \qquad b_t = b_{t-1} + \sigma^{-2} x_t y_t,$$

with posterior

$$p(\eta \mid \mathcal{H}_t) = \mathcal{N}\left(\Lambda_t^{-1} b_t, \Lambda_t^{-1}\right).$$

Before each execution, Thompson sampling draws $\tilde{\eta} \sim p(\eta \mid \mathcal{H}_t)$ for worker selection.

When a new worker joins, its probe signature immediately provides both $\hat{u}_w(s)$ and $g(w)$. No worker-specific online parameter is introduced, and the shared posterior over η remains unchanged.

B Experimental Details and Additional Results

B.1 Dataset Statistics.

Table 6 summarizes the datasets and splits used throughout our experiments. Note that the probe set contains 55,676 examples. Profiling a new worker therefore requires roughly 56K probe inferences, which is typically inexpensive in practice and is incurred only when the worker is introduced. Since worker updates are far less frequent than deployment queries, the profiling cost is amortized over the worker's subsequent lifetime and adds little recurring overhead.

B.2 Online Adaptation Split

For the online adaptation experiment, we randomly hold out examples from each evaluation benchmark to construct a fixed evaluation set. Because GAIA contains only 165 examples, we reserve the full benchmark for evaluation. The remaining examples form the adaptation stream used to update the contextual-bandit posterior.

Table 6: Data sources used for probe construction, planner post-training, and evaluation.

Usage	Dataset	Primary task type	# Examples	Sampling
Probe set	APPS	Code generation	4,962	Full
	BFCL	Tool / function use	3,151	Full
	Big-Math	Mathematical reasoning	6,000	Sampled
	FRAMES	Knowledge-intensive reasoning	824	Full
	MBPP+	Code generation	378	Full
	SuperGPQA	General knowledge / reasoning	26,529	Full
	MMLU-Pro	General knowledge / reasoning	12,032	Full
	ACPBench	Planning / agentic reasoning	1,800	Full
RL training	DAPO-Math-17K	Mathematical reasoning	1,000	Sampled
	MuSiQue	Multi-hop reasoning	675	Sampled
	BigCodeBench	Code generation	415	Sampled
	Logistics	Planning	400	Sampled
	CodeContests	Competitive programming	335	Sampled
	TACO	Code generation	335	Sampled
	BrowseComp-Plus	Retrieval / web reasoning	400	Sampled
	TravelPlanner	Planning	225	Sampled
Evaluation	MuSiQue	Multi-hop reasoning	2,000	Sampled
	TravelPlanner	Planning	1,000	Full
	Omni-MATH-2	Mathematical reasoning	4,428	Full
	LiveCodeBench v6	Competitive programming	611	Full
	BBEH	Broad reasoning	4,520	Full
	NaturalPlan	Constraint-based scheduling	3,600	Full
	PlanBench-mystery	Obfuscated symbolic planning	603	Full
	GAIA	General assistant / tool use	165	Full

Table 7: Dataset split for the online adaptation experiment. The held-out examples are used only for evaluation and never for posterior updates.

Benchmark	Total	Adaptation	Held-out
MuSiQue	2,000	1,750	250
TravelPlanner	1,000	800	200
Omni-MATH-2	4,428	4,128	300
LiveCodeBench v6	611	411	200
BBEH	4,520	4,220	300
NaturalPlan	3,600	3,300	300
PlanBench-mystery	603	391	212
GAIA	165	0	165
Total	16,927	15,000	1,927

B.3 Implementation Details

Planner training. We optimize the planner with a GRPO-style policy-gradient objective, applying A^{dec} to the Stage-I generation and A^{op} to the corresponding Stage-II generation, with the two losses weighted equally. We perform full-parameter RL post-training of the planner with AdamW using a learning rate of 1×10^{-6} under a cosine schedule with warmup ratio 0.03, a global batch size of 16 queries, and 400 optimization steps. For each training query, we sample $J = 8$ Stage-I decompositions and $k = 8$ Stage-II collaboration plans per decomposition, both at temperature 1.0. Group-relative advantages are mean-centered within each group and are not divided by the group standard deviation. We allow at most 8 subtasks per decomposition and cap the number of workers assigned to any single subtask at $K = 6$. The Stage-II cost coefficient is set to $\beta = 0.01$.

For planner post-training, terminal quality $Q(x, y)$ is computed by the task-specific verifier of each training source. For every source except TravelPlanner, the verifier is binary, with $Q \in \{0, 1\}$ and $Q = 1$ iff the generated solution passes. TravelPlanner instead uses a denser gated reward derived from the benchmark's own checkers: an invalid or undelivered plan receives zero reward, while a valid plan is scored as the equally weighted sum of five components—the fractions of commonsense and hard constraints satisfied, the corresponding two indicators of whether all such constraints are satisfied, and the benchmark's final all-or-nothing success indicator. We use this denser reward only for training because TravelPlanner's binary pass rate on our worker pool is near zero, causing rollout groups with uniformly zero rewards to provide no learning signal. Evaluation follows the standard task-specific grader of each benchmark, and all reported TravelPlanner results use the binary pass rate.

Static matcher. For probe-based matching, we use $h_{\mathrm{rel}} = 0.05$, $h_{\mathrm{den}} = 0.02$, $\nu = 1.0$, and $\epsilon = 0.01$. For MMR-based worker selection, we set the trade-off coefficient to $\lambda_o = 0.5$ for all multi-worker operations.

Online bandit. For online adaptation, we use the contextual Thompson-sampling formulation in Appendix A.1. We use the full 4096-dimensional `Qwen3-Embedding-8B` representation for both subtask embeddings $z(s)$ and probe embeddings $z(p_j)$. The shared online parameter η uses an isotropic Gaussian prior with precision 1.0, and the observation-noise variance is $\sigma^2 = 0.01$. Unless otherwise specified, all reported results use the static probe-based matcher without online updates; online adaptation is enabled only in the dedicated analysis in Table 5.

Unseen worker pool. For the worker-pool generalization experiment in Section 4.3, we replace the original six-worker pool with an entirely unseen pool of four models: `gemma-4-26B-A4B-it`, `Qwen3-14B`, `granite-4.2-8b`, and `Ministral-3-8B-Instruct-2512`. None of these workers is used during planner training. For DEORCH, each new worker is incorporated only by evaluating it on the fixed probe set; no planner or matcher retraining is performed. For CONDUCTOR, the four new workers are assigned the available worker indices and the orchestrator is informed that the current pool contains four workers, without further training. BEST SINGLE AGENT uses `gemma-4-26B-A4B-it`, the strongest worker in the new pool. INDEPENDENT executes all four workers independently and uses the strongest worker to aggregate their responses into the final answer. Because the current matcher forms teams from distinct workers, the default maximum team size $K = 6$ is clipped to the current pool size when fewer workers are available; the four-worker unseen pool therefore allows at most four workers per subtask. We omit MAS-Orchestra from the unseen-pool evaluation because it assumes a single underlying worker rather than selecting among a heterogeneous worker pool. Replacing this worker with one model from the new pool would test model substitution rather than adaptation to the pool, while introducing multi-worker selection would alter the original method.

Baseline adaptation. For CONDUCTOR (Nielsen et al., 2026), we replace its original worker pool with our six-worker pool and retrain the orchestrator following its original procedure. Although CONDUCTOR improves robustness to varying worker availability by training on randomized worker subsets, these subsets are drawn from the same fixed set of workers seen during training; its reported dynamic-pool evaluation therefore does not test transfer to entirely unseen workers. For MAS-ORCHESTRA (Ke et al., 2026), which uses a single underlying worker model, we replace that worker with `gemma-4-12B-it`, the strongest worker in our pool, while otherwise retaining its original

orchestration mechanism. To approximately match the training rollout budget of DEORCH ($400 \times 16 \times 8 \times 8 \approx 410$K executed orchestrations), we train CONDUCTOR for 1,600 optimization steps and MAS-ORCHESTRA for 200 steps, accounting for their different numbers of rollouts per step.

All experiments are conducted on a single node with 4× NVIDIA H200 GPUs. Code is available at: https://anonymous.4open.science/r/DeOrch-code-F8D3.

B.4 OFFLINE MATCHER EVALUATION PROTOCOL

For the in-distribution evaluation of the static matcher, examples from each probe source are randomly split 80/20, with 80% used to construct the probe set and the remaining 20% held out for evaluation. For leave-one-source-out (LOSO) evaluation, we consider five representative sources—ACPBench, MMLU-Pro, MBPP+, Big-Math, and BFCL—and exclude the corresponding source entirely from the probe set before evaluating on its held-out examples.

We compare against random routing, the strongest individual worker in the pool (`gemma-4-12B-it`), majority voting over all six workers, and probe matching without density correction. We additionally report ORACLE SINGLE, which selects a successful worker for each example in hindsight and serves only as an upper bound.

B.5 COMPLEMENTARITY-AWARE TEAM SELECTION.

The previous experiment evaluates single-worker selection, while multi-worker operations additionally require constructing worker teams. To test whether probe-derived complementarity improves team selection, we construct a TOP-n SUITABILITY variant of DEORCH. Using the same planner outputs and team sizes as the full method, this variant selects the top-n workers solely according to their suitability scores $\hat{u}_w(s)$, without using complementarity information. Table 8 shows that complementarity-aware selection consistently improves over selecting workers solely by individual suitability. Since both variants execute the same plans with the same team sizes, the improvement cannot be attributed to additional worker calls, but instead reflects the benefit of accounting for worker complementarity when constructing multi-worker teams.

Table 8: Inference-time ablation of multi-worker selection.

Selection Rule	ID Avg. ↑	OOD Avg. ↑	Overall Avg. ↑
TOP-n SUITABILITY	45.9	52.3	50.7
Complementarity-Aware selection	47.4	52.9	51.5

B.6 INFERENCE COST ANALYSIS

We further evaluate inference cost by counting total prompt and completion tokens across all autoregressive model calls, including planning, worker execution, and collaboration. Token usage is averaged within each benchmark and macro-averaged across the eight benchmarks. We additionally include BEST SINGLE + REFINEMENT to control for increased inference compute: starting from `gemma-4-12B-it`, it performs three sequential refinement rounds without ground-truth feedback, yielding four worker calls and a token budget comparable to DEORCH.

Table 9 shows that refinement improves BEST SINGLE AGENT from 39.2 to 42.4, but remains well below DEORCH's 51.5 at a similar token budget (14.1K vs. 14.6K). DEORCH also outperforms INDEPENDENT, CONDUCTOR, and MAS-ORCHESTRA while using fewer tokens, indicating that its gains are not explained by increased inference compute alone.

B.7 COMPARISON WITH COUNTERFACTUAL CREDIT ASSIGNMENT

LEMON addresses orchestration credit assignment through counterfactual edits to its generated orchestration. We do not include it as a direct baseline in Table 1 because LEMON assigns each role to one of three predefined capacity levels (*small*, *medium*, or *large*), whereas our setting requires selecting freely among six heterogeneous workers. Adapting LEMON to this setting would require

Table 9: Inference cost across the eight evaluation benchmarks.

Method	Avg. ↑	Calls ↓	Tokens / Query ↓
BEST SINGLE AGENT	39.2	1.00	1859
BEST SINGLE + REFINEMENT	42.4	4.00	14106
INDEPENDENT	46.0	7.00	25859
FRONTIER-PLANNER	36.6	4.67	11479
UNTRAINED PLANNER	38.3	4.92	12867
CONDUCTOR	44.0	6.16	22458
MAS-ORCHESTRA	49.1	5.63	17250
DEORCH	51.5	5.35	14577

redefining both its worker-selection representation and the corresponding counterfactual edits, making the comparison no longer a faithful reproduction of the original method.

We instead construct a controlled LEMON-inspired counterfactual-credit baseline in our orchestration space. This variant replaces our two-stage planner and conditional-credit objective with single-pass orchestration trained using LEMON-style global and localized counterfactual objectives, while keeping the worker pool, matcher, execution protocol, and training budget unchanged. Specifically, following LEMON's counterfactual training procedure, we train the single-pass planner with global GRPO augmented by one localized counterfactual edit per rollout; we adapt its mutations to our orchestration space by editing either a decomposition dependency or a collaboration operation, re-execute the modified plan, and apply the resulting reward contrast only to the edited token span. This directly compares two approaches to credit assignment across decomposition and operation selection: counterfactual credit assignment versus our two-stage conditional formulation. As shown in Table 10, DEORCH improves the overall average from 49.2 to 51.5 with comparable worker calls, indicating that our two-stage conditional formulation provides more effective credit assignment in this setting.

Table 10: Comparison of credit-assignment formulations.

Variant	ID Avg.	OOD Avg.	Avg.	Calls ↓
Counterfactual Credit	43.1	51.2	49.2	5.17
Two-Stage Conditional Credit (DEORCH)	47.4	52.9	51.5	5.35

B.8 DISCUSSION AND FUTURE WORK

Computational overhead. Compared with single-pass planning, our two-stage planner introduces one additional planner generation at inference time. This overhead is modest relative to the subsequent execution of multiple worker agents; indeed, our cost analysis shows that DEORCH still uses fewer inference tokens than competing learned MAS orchestrators. Training incurs a larger overhead because the nested conditional rollout procedure samples multiple Stage-II plans for each sampled decomposition. However, this additional training cost is incurred only during post-training: once trained, the same planner can be reused when the worker pool changes without further retraining. Improving the efficiency of this nested training procedure is an interesting direction for future work.

Extending the collaboration space. Our current multi-worker operations construct teams from distinct workers in the pool and therefore do not support homogeneous collaboration through repeated execution of the same worker. In practice, an ENSEMBLE-3 operation could either invoke three distinct workers or run three independent instances of the same strong worker, and which choice is preferable may depend on the task. A natural solution is to introduce both homogeneous and heterogeneous variants into the operation vocabulary, such as homogeneous-ENSEMBLE-3 and heterogeneous-ENSEMBLE-3, allowing the planner to learn which form of collaboration is more appropriate for each subtask. We also considered richer operations that are not explored in this work due to computational constraints. A recursive operation could further decompose a difficult subtask during execution, while an escalation operation could invoke an external worker outside the current

pool when the available workers are insufficient. Incorporating such operations would extend the framework toward more adaptive and open-ended orchestration.

Scaling to larger worker pools. Our experiments consider heterogeneous pools of up to six workers. Although the planner interface is independent of worker identities and pool size, the behavior of the matcher at substantially larger scales remains to be evaluated. Evaluating and improving DEORCH under substantially larger and more dynamic worker pools is therefore an important direction for future work.