LEARNING TO ORCHESTRATE WITH COUNTERFACTUAL PREFERENCES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) orchestration requires jointly deciding which workers to call, what subtasks to assign, and how to pass information between them. Terminal task rewards make this hard to learn, since one outcome must be credited across many coupled choices. We introduce CPO-ORCH, an offline preference-learning approach that localizes this supervision. It constructs *decision-localized counterfactuals* by editing exactly one decision in an executed workflow (a worker, a subtask instruction, or an access edge), re-executing both variants, and retaining pairs whose outcomes differ consistently under repeated execution. It combines these with natural preference pairs, structurally matched to suppress worker-identity shortcuts, and trains an 8B orchestrator with direct preference optimization, without online rollouts. Across four reasoning domains, CPO-ORCH reaches 85.56% macro accuracy, outperforming the strongest fixed worker in its pool (83.18%), the strongest single model evaluated outside the pool (82.97%), the best fixed multi-agent scaffold (84.02%), and an oracle that selects the best worker per domain, while ablations show that natural and counterfactual pairs are complementary.

1 INTRODUCTION

Modern LLM systems increasingly rely on multiple heterogeneous models rather than a single universal model. The central challenge is therefore shifting from *which model is strongest* to *how models should be composed*. A learned orchestrator must jointly decide which workers to invoke, what instructions to give them, and how information should flow between calls. The RL Conductor demonstrates that a small language model can learn such natural-language workflows from end-to-end outcome rewards (Nielsen et al., 2026). However, a terminal success signal assigns the same scalar outcome to an entire structured workflow, leaving the learning algorithm to infer which of many coupled decisions actually mattered.

We ask a different question: *what supervision teaches an orchestrator which decisions are consequential?* We construct preferences from two sources. *Natural pairs* match successful and failed workflows of the same task, drawn from rollouts produced during GRPO training of the RL Conductor baseline and filtered by structural constraints (Fig. 2A). Such pairs usually differ in many decisions at once. *Counterfactual pairs* instead intervene on the workflow itself: we modify exactly one primary orchestration decision (e.g., the final worker, a non-final worker, a synthesis instruction, a subtask, or an information-access edge) and re-execute both the source and edited workflows (Fig. 2B). When the outcomes differ, the pair provides a localized preference over one choice while holding the rest of the workflow largely fixed. We call these pairs *counterfactual* only in this operational sense: because worker execution is stochastic, an outcome difference need not be caused solely by the edited decision.

We instantiate this idea as CPO-ORCH, a Qwen3-8B orchestrator trained with offline direct preference optimization (DPO) on natural and counterfactual preference pairs. We evaluate it against fixed workers, fixed and random multi-agent scaffolds, oracle routing bounds, and the RL Conductor baseline and its warm-started GRPO extension, all under a common evaluation protocol, with orchestrators decoded greedily (temperature 0) and workers decoded at temperature 0.2, on a four-domain, 198-task evaluation set. Our findings are threefold.

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

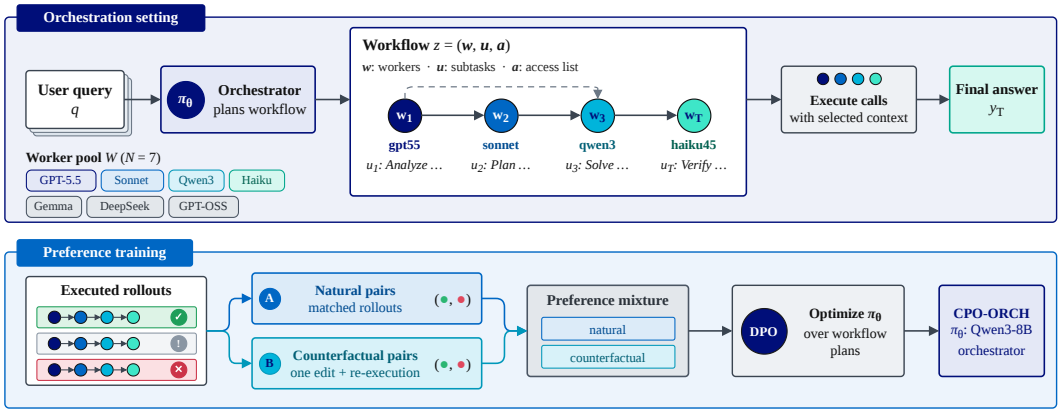

Figure 1: Overview of CPO-ORCH. **Top:** a Qwen3-8B orchestrator plans a workflow of worker calls, subtasks, and access lists over a fixed heterogeneous worker pool. **Bottom:** from executed rollouts we build natural preferences from matched rollouts and decision-localized counterfactual preferences by editing and re-executing one workflow decision (Fig. 2), then optimize the orchestrator offline with DPO.

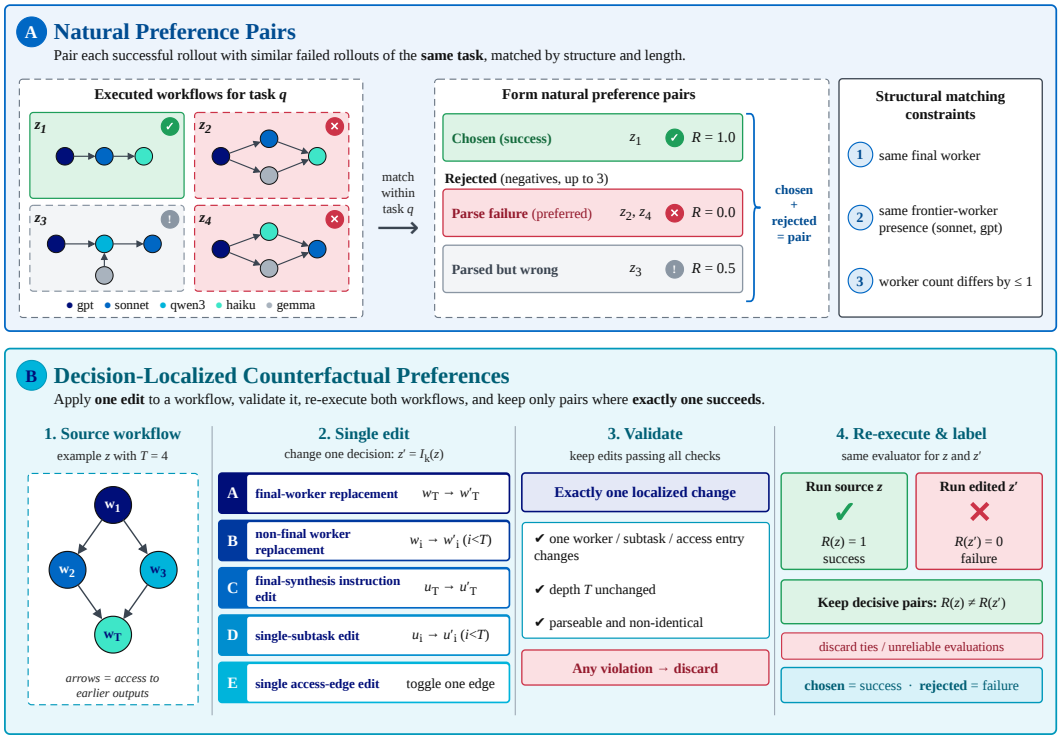

Figure 2: Preference-pair construction. **(A) Natural pairs:** each successful rollout is paired with failed rollouts of the same task, subject to structural matching constraints (Sec. 3.2). **(B) Counterfactual pairs:** one decision of an executed workflow is edited by one of five operators, the edit is validated, both workflows are re-executed, and only pairs in which exactly one succeeds are kept (Sec. 3.3).

First, *offline preference optimization alone learns strong orchestration policies*. CPO-ORCH reaches 85.56% macro accuracy, outperforming the strongest fixed worker in the pool (`sonnet46`, 83.18%), the strongest single model evaluated outside the pool (82.97%), and the best fixed multi-agent scaffold (84.02%). A GRPO stage initialized from the CPO-ORCH checkpoint reaches 84.58%, providing no additional improvement over the offline method.

Second, *natural and counterfactual supervision are complementary*. Under a fixed pair-exposure budget, an even mixture of the two sources outperforms both natural-only and counterfactual-only training, while other mixing ratios remain within a narrow range, so the benefit does not hinge on tuning the ratio precisely. Likewise, counterfactuals that modify worker assignments perform similarly to those that modify synthesis instructions, subtasks, or information-access edges, and combining both edit families is numerically strongest. The supervision therefore does not reduce to learning worker selection alone.

Third, *the gains go beyond coarse routing*. An oracle that assigns the empirically best single worker to each domain reaches 83.77%, below CPO-ORCH despite using evaluation-set labels. At the same time, an oracle best-of-pool diagnostic reaches 91.44%, revealing substantial remaining per-task complementarity across workers. Consistent with this interpretation, orchestration gains largely disappear on a temporally held-out coding benchmark whose challenging tasks lie beyond the capabilities of the current worker pool. The learned behavior is also stable across independent worker re-executions rather than being driven by a single favorable sample.

Together, these observations motivate a view of orchestration learning in which global workflow examples provide a prior over plausible plans, while localized interventions reveal which individual decisions matter. Our contributions are:

- A *counterfactual preference construction* for orchestration: single-decision workflow edits, re-executed and retained only when the task outcome changes, yielding localized preferences over individual orchestration choices (Sec. 3).
- Evidence that *offline* preference optimization with such data trains a Qwen3-8B orchestrator that outperforms every fixed worker evaluated, the best fixed multi-agent scaffold, and learned and heuristic single-worker routers, and exceeds even an oracle per-domain router, while an additional online RL stage provides no improvement (Secs. 4.2 and 4.4).
- Controlled ablations showing that mixing natural and counterfactual preferences outperforms either source alone and that the benefit does not depend specifically on worker-selection edits (Secs. 4.3 and 4.4).

2 RELATED WORK

Learned orchestration. Recent work treats the agentic system itself as an object to optimize: GPTSwarm optimizes prompts and connectivity of agent graphs (Zhuge et al., 2024), ADAS lets a meta-agent write and evaluate new agent programs (Hu et al., 2025), and AFlow searches over code-represented workflows with Monte Carlo tree search (Zhang et al., 2025b). Other methods adapt the communication structure through dynamic agent-team selection (Liu et al., 2023), task-dependent communication graphs (Zhang et al., 2025a; Leong et al., 2025), or reinforcement-learned centralized orchestration (Dang et al., 2025). These approaches typically optimize a graph, program, or constrained controller rather than individual decisions inside a free-form natural-language workflow. The closest setting to ours is the RL Conductor (Nielsen et al., 2026), a small language model that emits complete natural-language orchestration plans and is trained from the correctness of the executed workflow. We adopt its action space but study a different learning signal: our contribution is a data-construction mechanism that localizes supervision to the decisions inside a workflow, not a new orchestration representation.

Preference optimization and preference-data construction. DPO turns pairwise preferences into a policy objective without fitting an explicit reward model or running online RL (Rafailov et al., 2023). Its effectiveness depends strongly on how comparisons are collected: on-policy and suboptimal samples can be valuable (Tajwar et al., 2024), reward margins affect pair selection (Huang et al., 2025), and observational preferences can entangle the factors behind a preference (Kobalczyk & Van Der Schaar, 2025). This entanglement is acute for orchestration, where a successful and a failed workflow for the same task may differ simultaneously in workers, depth, instructions, and communication. We therefore leave the DPO objective unchanged and instead construct comparisons whose granularity matches the decisions the orchestrator must learn: structurally matched natural pairs, complemented by interventions on exactly one decision.

Counterfactual and interventional supervision. Counterfactually-augmented data minimally edit an input to flip its label, encouraging reliance on causally relevant features (Kaushik et al., 2020). Related work constructs counterfactuals through task-relevant edits (Zhang et al., 2025c), actively queries counterfactual trajectories for reward inference (Tung et al., 2026), or builds counterfactual reasoning traces for preference-based unlearning (Li et al., 2026). None of these edit a single decision in an executable heterogeneous-worker workflow and let re-execution supply the preference label. We use *counterfactual* in this operational sense: because workers are stochastic, a re-executed pair gives a localized empirical comparison rather than an exact causal effect.

Positioning. The Conductor (Nielsen et al., 2026) shares our free-form orchestration object and action space but learns from a single terminal reward via online RL; we instead learn offline from localized, execution-verified contrasts. Counterfactually-augmented data (Kaushik et al., 2020) share our single-factor-edit logic but edit static inputs without execution; we edit one decision inside an executable workflow and let re-execution supply the label. To our knowledge, prior work does not combine decision-localized, execution-grounded counterfactual preferences over a free-form orchestration workflow with offline optimization. Appendix A further discusses model routing and heterogeneous worker selection (Ong et al., 2024; Yue et al., 2025), including how our setting differs from choosing a single model per query, as well as hand-designed multi-agent scaffolds and credit assignment for multi-step reasoning.

3 METHOD

We train a small orchestration policy π_θ to coordinate a fixed pool of heterogeneous LLM workers. Our method, CPO-ORCH, is an *offline preference optimization* approach: rather than learning exclusively from terminal rewards, we construct preference pairs that localize supervision to individual orchestration decisions and optimize the orchestrator with direct preference optimization (DPO). The main contribution is therefore the preference *data construction*, not a new preference-learning objective.

Our point of reference is the Conductor formulation of Nielsen et al. (2026), in which a small language model emits natural-language multi-worker workflows and is trained from executed task rewards using online RL. We retain the same basic orchestration interface and action space, but instead learn primarily from offline natural and counterfactual workflow preferences. We additionally study online RL as an optional second stage in Sec. 3.6, while it is not part of CPO-ORCH.

3.1 ORCHESTRATION FORMULATION

Given a user problem x, the orchestration policy π_θ emits an ordered workflow

$$z = \big((w_1, u_1, a_1), \ldots, (w_T, u_T, a_T)\big),$$

where $w_t \in \{1, \ldots, N\}$ denotes the worker assigned to step t, u_t is a natural-language subtask instruction, and $a_t \subseteq \{1, \ldots, t-1\}$ is an access list specifying which earlier worker outputs step t may condition on. Workers execute sequentially, and the final worker produces the system response y_T. Although execution is sequential, the access lists define a directed acyclic graph over steps, so a single format expresses diverse workflow topologies: a chain when each step accesses its predecessor, independent parallel attempts when access lists are empty, and aggregation or verification when a later step accesses several earlier outputs. The orchestrator prompt template, worker descriptions, and in-context examples, which follow Nielsen et al. (2026) and identify workers only by ordinal descriptions, are documented in Appendix L.

We write a workflow as

$$z = (\mathbf{w}, \mathbf{u}, \mathbf{a}),$$

where $\mathbf{w}$ is the worker sequence, $\mathbf{u}$ the sequence of subtask instructions, and $\mathbf{a}$ the information-access structure. The final response is evaluated by a task-specific binary evaluator

$$R(x, y_T) \in \{0, 1\}.$$

Our experiments use a fixed pool of $N = 7$ heterogeneous workers and a Qwen3-8B orchestration policy. Implementation and decoding details are given in Sec. 4.1.

3.2 NATURAL PREFERENCE PAIRS

We first mine preference pairs from previously executed orchestration rollouts (Fig. 2A). For a given task, successful workflows are treated as chosen examples and unsuccessful workflows as rejected examples. The original rollout reward distinguishes three outcomes: 1.0 for a parsed and correct workflow, 0.5 for a parsed but incorrect workflow, and 0.0 for a parse failure.

Within each task, a successful workflow is paired with structurally similar rejected workflows. We prioritize parse-failure negatives over parsed-but-incorrect negatives and match each chosen workflow to nearby rejected workflows according to serialized workflow length. We retain at most three rejected workflows per chosen workflow and cap the number of pairs contributed by any one task. For RLPR, we retain only negatives whose failure status is unambiguous under the available labeling procedure. For code-generation tasks, correctness labels are obtained by executing the generated program against the task's test cases.

Successful workflows disproportionately call the two strongest workers in the pool, `sonnet46` and `gpt55` (Table 4), and training on unmatched pairs tends to collapse the learned workflows onto them, reducing orchestration to a "pick the strongest worker" heuristic. We therefore impose structural matching. For a natural pair (z^+, z^-), the chosen and rejected workflows must

1. use the same final worker;
2. agree on whether each of these two strongest workers appears anywhere in the workflow;
3. differ in workflow depth by at most one step.

Formally,

$$\begin{gathered}\text{final}(z^+) = \text{final}(z^-),\\ \mathbf{1}[m \in z^+] = \mathbf{1}[m \in z^-] \qquad \forall m \in \{\texttt{sonnet46}, \texttt{gpt55}\},\end{gathered}$$

and

$$|T^+ - T^-| \leq 1.$$

This matching suppresses preferences driven primarily by the identity of a dominant final worker and shifts the supervision toward differences in workflow structure and composition. The resulting natural-preference pool contains 2602 matched pairs over 662 tasks.

3.3 DECISION-LOCALIZED COUNTERFACTUAL PREFERENCES

Natural preferences compare workflows that happened to occur during rollout collection. To obtain more localized supervision, we instead intervene directly on an executed workflow (Fig. 2B).

Given a source workflow

$$z = (\mathbf{w}, \mathbf{u}, \mathbf{a}),$$

we construct a counterfactual workflow

$$z' = \mathcal{I}_k(z)$$

by applying exactly one edit operator $\mathcal{I}_k$. We use five operator families:

(A) **Final-worker replacement.** Replace the final worker w_T with a different worker w'_T.

(B) **Non-final-worker replacement.** Replace one worker w_i with w'_i for some $i < T$.

(C) **Final-synthesis-instruction edit.** Replace the final synthesis instruction u_T with an alternative from a fixed template family.

(D) **Single-subtask edit.** Modify one non-final subtask instruction u_i using a fixed intent-preserving edit template.

(E) **Single-access-edge edit.** Add or remove exactly one dependency in a step's access list.

The edit operators are rule-based rather than model-generated. Worker replacements choose a distinct worker according to a fixed balancing rule, while instruction edits are drawn deterministically from frozen template sets (Appendix L). Thus the counterfactual generator introduces no additional language-model sampling.

Strict single-decision validation. Every candidate pair (z, z') is passed through a structural validator that enforces that exactly one primary orchestration decision has changed. The edit must preserve the workflow depth T, and across the three decision fields $\{\texttt{model_id}, \texttt{subtasks}, \texttt{access_list}\}$ the total number of changed positions must equal one:

$$\big|\{t : w_t \neq w'_t\}\big| + \big|\{t : u_t \neq u'_t\}\big| + \big|\{t : a_t \neq a'_t\}\big| = 1.$$

Candidates are rejected if they are unparseable, identical to the source workflow, change the number of steps, or modify more than one decision. Each retained candidate therefore differs from its source by one localized orchestration intervention. Because later workers may condition on earlier outputs, one changed decision can still alter what downstream steps see; a pair thus isolates one *decision*, whose effect includes these downstream consequences.

Re-execution and preference labeling. We do not infer a preference from the source workflow's cached outcome. Instead, both the source workflow z and its edited counterpart z' are executed again under the same worker pool and evaluation procedure. We retain only *decisive* comparisons in which exactly one workflow succeeds:

$$R(x, y_T) = 1, \qquad R(x, y'_T) = 0,$$

or vice versa.

The successful workflow becomes the chosen response and the unsuccessful workflow the rejected response. The edited workflow may therefore be either preferred or rejected; an example pair is shown in Appendix M. Pairs in which both workflows succeed, both fail, or cannot be evaluated reliably are discarded.

This construction differs from ordinary preference mining in an important way: the two workflows are intentionally matched everywhere except for one orchestration decision, while the resulting preference is determined by actual execution.

Balancing and repeated-execution curation. The accepted counterfactual pool is balanced to prevent a single edit type, domain, or worker-replacement direction from dominating the training data. We cap the contribution of any individual operator at 35%, any domain at 45%, and any worker-replacement direction at 30%.

Worker execution is stochastic, so a preference observed in one pair of executions need not reflect a stable difference between the two workflows. In preliminary experiments on an earlier setup, only about a quarter of single-execution preferences reproduced under re-execution, and validating them improved macro accuracy by about 3 points under a matched recipe. We therefore further curate the counterfactual pool through repeated execution: each source–counterfactual pair is executed five times, and a pair is retained only if at least four of the five paired executions yield the same preference direction. This curation is part of data construction, not a separate learning objective.

3.4 Combining natural and counterfactual supervision

The natural and counterfactual preference sources provide complementary supervision. Natural pairs compare structurally matched workflows that arose organically during rollout collection. Counterfactual pairs instead intervene on one specific workflow decision while holding the remainder of the workflow fixed.

We train CPO-Orch on a mixture of the two sources. The precise construction of the final training mixture, together with controlled ablations over the natural–counterfactual ratio, is described in Sec. 4.1 and Sec. 4.3. Keeping these data-mixture choices separate from the method definition allows us to study whether performance depends on the amount or type of counterfactual supervision without changing the underlying construction.

3.5 Offline preference optimization

For each prompt x, a preference example consists of a chosen workflow z^+ and a rejected workflow z^-. We optimize the orchestration policy with DPO (Rafailov et al., 2023) relative to a frozen

reference policy π_{ref}:

$$\mathcal{L}_{\text{DPO}} = -\log \sigma\left(\beta\left[\log \frac{\pi_\theta(z^+ \mid x)}{\pi_{\text{ref}}(z^+ \mid x)} - \log \frac{\pi_\theta(z^- \mid x)}{\pi_{\text{ref}}(z^- \mid x)}\right]\right).$$

The likelihood is computed over the serialized orchestration plan rather than over worker-generated task answers. Thus DPO directly trains the orchestrator to prefer one workflow construction over another. The reference policy is the preference-untrained orchestration policy. Optimization hyperparameters and parameter-efficient fine-tuning details are reported in Sec. 4.1.

3.6 Optional online-RL extension

Online RL is not part of CPO-Orch, but we include it both as a point of comparison and as an optional second training stage.

The GRPO-only baseline follows the prior Conductor setting and initializes from the preference-untrained policy. The warm-start variant instead initializes from the DPO-trained CPO-Orch checkpoint. Apart from initialization, the two online-RL runs use the same training procedure, allowing the comparison to isolate the effect of the offline preference stage.

For each prompt, GRPO samples a group of G workflows, executes them, and assigns task rewards R_i. The group-standardized reward is used as the advantage:

$$A_i = \frac{R_i - \text{mean}_j R_j}{\text{std}_j R_j + \epsilon}.$$

Rollouts are collected synchronously and on-policy, so the importance ratio is one and each update reduces to an advantage-weighted policy gradient; the optimization schedule and rollout budget are given in Appendix I.

4 Experiments

4.1 Setup

We evaluate on a four-domain, 198-task evaluation set (mathematics 60, MMLU 60, RLPR 60, LiveCodeBench 18) with a fixed pool of seven heterogeneous LLM workers. Workers are decoded at temperature 0.2 and the orchestrator is decoded greedily (temperature 0) unless stated otherwise. We report *macro* accuracy, the unweighted mean of the four domain accuracies, averaged over three worker-sampling seeds. The orchestrator backbone is Qwen3-8B; the worker pool is fixed across all orchestration systems (Appendix B).

CPO-Orch is trained with DPO on natural preferences and repeated-execution–curated counterfactual preferences constructed as in Sec. 3. Unless stated otherwise, the two sources are mixed evenly ($\rho = 0.5$). The mixing-ratio study uses matched 2034-pair arms while varying $\rho \in \{0, 0.25, 0.40, 0.50, 0.60, 0.75, 1.00\}$. The operator study uses matched arms with 1500 shared natural pairs and 534 counterfactual pairs. Every trained baseline, including Natural DPO, was tuned separately and is reported at its best configuration. Full settings are in Appendix I.

4.2 Main comparison

CPO-Orch reaches 85.56% macro accuracy, outperforming the best fixed multi-agent scaffold (84.02%), the strongest fixed worker in the pool (`sonnet46`, 83.18%), and the strongest single model evaluated outside the pool (`fable5`, 82.97%). Because CPO-Orch can call only the seven in-pool workers, its gain cannot be attributed to access to a stronger external model.

4.3 How much counterfactual data is needed?

We vary the counterfactual share ρ in matched 2034-pair training arms, with three training seeds per setting and three worker-sampling draws per checkpoint (Table 2). These arms use a lighter ablation setting than the main CPO-Orch configuration, so we compare them only with one another.

Table 1: Main comparison on the 198-task evaluation set (macro; greedy orchestrator decoding; three worker-sampling draws; per-domain rates shown). Macro is computed from unrounded domain accuracies. ‡ out-of-pool reference. † non-learnable oracle diagnostics using evaluation-set outcomes.

System	Macro	math	mmlu	rlpr	LCB
Pool worker (`gpt55`)	82.27	75.6	97.2	71.1	85.2
Strongest fixed worker in-pool (`sonnet46`)	83.18	79.4	97.2	72.8	83.3
Strongest single model out-of-pool (`fable5`)‡	82.97	81.1	90.0	75.6	85.2
Best fixed multi-agent scaffold (`gpt55`→`sonnet46`)	84.02	78.3	95.6	73.3	88.9
Train-domain router (heuristic)	82.92	78.3	98.3	71.7	83.3
Learned single-worker router	75.47	77.2	86.7	63.9	74.1
Compute-matched random workflow	67.66	57.8	92.8	47.8	72.2
Policy base (untrained)	77.13	78.3	86.7	63.9	79.6
Natural DPO	81.52	83.3	95.0	70.0	77.8
Conductor (GRPO-only, cold)	79.25	79.4	89.4	66.7	81.5
CPO-ORCH → GRPO (warm)	84.58	84.4	96.7	73.9	83.3
CPO-ORCH	**85.56**	83.3	98.3	71.7	88.9
Oracle per-domain router†	83.77	78.3	98.3	73.3	85.2
Oracle best-of-pool†	91.44	90.0	98.3	86.7	90.7

Table 2: Counterfactual share ρ versus macro accuracy in the controlled mixing sweep. All arms use 2034 unique pairs sampled without replacement and the same ablation training setting, which differs from the main CPO-ORCH configuration (Appendix I); absolute accuracies are therefore not comparable to Table 1. The main CPO-ORCH result is shown below for reference.

ρ (counterfactual share)	Macro	math	mmlu	rlpr	LCB
0.00	82.61	82.6	96.1	67.8	84.0
0.25	82.38	81.7	97.2	66.7	83.9
0.40	83.19	81.3	97.8	68.5	85.2
0.50	**84.06**	82.8	96.9	70.2	86.4
0.60	82.21	81.3	95.6	67.4	84.6
0.75	82.13	82.0	95.4	69.6	81.5
1.00	81.91	80.4	92.8	70.6	84.0
Main CPO-ORCH configuration ($\rho = 0.5$)	85.56	83.3	98.3	71.7	88.9

The even mixture performs best (84.06%), 1.45 points above natural-only and 2.15 points above counterfactual-only training, while the remaining ratios lie within about 1.3 points of one another. Combining the two sources thus helps, without requiring precise tuning of ρ. Since changing ρ also changes task and domain composition (Appendix E), these differences are not a pure effect of the ratio.

4.4 ANALYSIS

Which counterfactual edits matter? We compare matched training arms whose counterfactual component contains only worker-routing edits, only non-routing edits (instruction, subtask, or access-edge changes), or both. Their macro accuracies are 82.61%, 82.92%, and 83.39%, respectively. Both edit families therefore provide useful supervision, and the mixed arm is numerically strongest, suggesting that routing and non-routing counterfactuals may contribute complementary signals. These matched arms use a smaller, fixed training mixture, so the 83.39% mixed arm is not intended to reproduce the full CPO-ORCH score of 85.56%. Details are in Appendix G.

Beyond domain-level routing. Two oracle diagnostics clarify what routing alone can explain. An oracle that assigns the empirically best single worker to each domain reaches 83.77%, whereas CPO-ORCH reaches 85.56% without evaluation-set labels. An oracle that accepts any pool worker's

correct answer on each task reaches 91.44%, and this remaining gap is concentrated in RLPR and mathematics. Thus the gain cannot be reduced to choosing one fixed worker per domain. Per-task workflow forensics and repeat-execution analysis are reported in Appendix H, with example workflows in Appendix M.

When tasks exceed the pool's capabilities. Orchestration can exploit capabilities present in the worker pool but cannot create capabilities beyond them, so its benefit is bounded on sufficiently challenging tasks. On a temporally held-out 144-task LiveCodeBench set, two separately trained orchestration policies reach 35.19% and 34.03%, while across both systems and all three seeds only $59/144$ of these challenging tasks (40.97%) are solved by any worker. In contrast, on the main LiveCodeBench slice, CPO-ORCH reaches 88.9% against 79.6% for the untrained policy. Thus low worker coverage sharply limits the room for orchestration to help.

Online RL extension. Cold-start GRPO reaches 79.25%, while the same online-RL procedure initialized from CPO-ORCH reaches 84.58%. The 5.33-point difference isolates the value of the offline preference-trained initialization. The warm-start run shows no clear improvement over the offline checkpoint (84.58% versus 85.56%), possibly because the offline policy is already near what an 8B orchestrator can achieve with this pool. Since each GRPO run also requires $19{,}200$ online rollouts and about $30\times$ the GPU time of DPO, which issues no worker calls during optimization, we use the offline checkpoint as the final method. Appendix I compares the training costs.

4.5 INFERENCE COST AND WORKER CALLS

CPO-ORCH uses 2.21 worker calls per task, essentially matching the random workflow (2.22), yet reaches 85.56% rather than 67.66% macro accuracy (Table 3). It also incurs about 14% lower measured API cost than the best fixed scaffold while improving accuracy by 1.54 points. Thus the gain is not explained by more calls or higher inference spending, but by how the orchestrator allocates and composes those calls; detailed token statistics are in Appendix J.

Table 3: Worker calls per task on the 198-task evaluation set. The random-workflow control is matched to CPO-ORCH's call volume by construction.

System	Calls/task	Macro
Strongest fixed worker in-pool (`sonnet46`)	1.00	83.18
Best fixed multi-agent scaffold (`gpt55`→`sonnet46`)	2.00	84.02
Compute-matched random workflow	2.22	67.66
CPO-ORCH	2.21	**85.56**

5 DISCUSSION AND LIMITATIONS

Our results suggest that for orchestration, *how supervision is constructed* matters more than how it is optimized: one offline DPO stage on well-constructed pairs outperforms outcome-only online RL at a fraction of its cost. Natural pairs provide a prior over plausible workflows, while counterfactual pairs reveal which decisions change the outcome, and mixing them beats either alone. Since the gains come from composition and are bounded by what the pool can solve, promising directions include assembling pools for per-task complementarity and adding post-execution verification or answer selection to narrow the remaining gap to oracle best-of-pool.

Several limitations remain. Worker execution is stochastic, so re-executed pairs give localized empirical preferences rather than deterministic causal effects; the contribution of repeated-execution curation was measured only in preliminary experiments on an earlier setup. We use a single Qwen3-8B orchestrator and one seven-worker pool, and generalization to other backbones or pools remains open. In the mixing study, the two pools differ in task and domain coverage, so changing ρ also shifts the training distribution. Finally, our analysis of when orchestration cannot help is limited to a single coding benchmark.

AI USE STATEMENT

Generative AI tools were used to assist with literature discovery, manuscript editing and polishing. The authors reviewed and verified the cited literature, experimental results, numerical values, and scientific claims, and take full responsibility for the content of the submission.

REFERENCES

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Step-level value preference optimization for mathematical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7889–7903. Association for Computational Linguistics, 2024a. doi: 10.18653/v1/2024.findings-emnlp.463. URL https://aclanthology.org/2024.findings-emnlp.463/.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023. URL https://arxiv.org/abs/2305.05176.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, 2024b.

Yu-Neng Chuang, Prathusha Kameswara Sarma, Parikshit Gopalan, John Boccio, Sara Bolouki, Xia Hu, and Helen Zhou. Learning to route LLMs with confidence tokens. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 10859–10878. PMLR, 2025.

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. *arXiv preprint arXiv:2505.19591*, 2025. URL https://arxiv.org/abs/2505.19591.

Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian Del Carmen Hipolito Garcia, Menglin Xia, Laks V. S. Lakshmanan, Qingyun Wu, and Victor Rühle. BEST-route: Adaptive LLM routing with test-time optimal compute. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 13870–13884. PMLR, 2025.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 2024.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024. URL https://arxiv.org/abs/2402.01680.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, 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 *International Conference on Learning Representations*, 2024.

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

Kexin Huang, Junkang Wu, Ziqian Chen, Xue Wang, Jinyang Gao, Bolin Ding, Jiancan Wu, Xiangnan He, and Xiang Wang. Larger or smaller reward margins to select preferences for LLM alignment? In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 25922–25946. PMLR, 2025.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-Blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2023.

Divyansh Kaushik, Eduard Hovy, and Zachary C. Lipton. Learning the difference that makes a difference with counterfactually-augmented data. In *International Conference on Learning Representations*, 2020.

Kasia Kobalczyk and Mihaela Van Der Schaar. Preference learning for AI alignment: a causal perspective. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 31063–31083. PMLR, 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. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-industry.144. URL https://aclanthology.org/2025.emnlp-industry.144/.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Junyi Li, Yongqiang Chen, and Ningning Ding. CiPO: Counterfactual unlearning for large reasoning models through iterative preference optimization. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3152–3170. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.143. URL https://aclanthology.org/2026.acl-long.143/.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri 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*, 2024.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. Dynamic LLM-agent network: An LLM-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*, 2023. URL https://arxiv.org/abs/2310.02170.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023.

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*, 2026. URL https://arxiv.org/abs/2512.04388. arXiv:2512.04388.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs with preference data. *arXiv preprint arXiv:2406.18665*, 2024. URL https://arxiv.org/abs/2406.18665.

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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741, 2023.

Chenyang Shao, Xinyang Liu, Yutang Lin, Fengli Xu, and Yong Li. Route-and-reason: Scaling large language model reasoning with reinforced model router. *arXiv preprint arXiv:2506.05901*, 2025. URL https://arxiv.org/abs/2506.05901.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Fahim Tajwar, Anikait Singh, Archit Sharma, Rafael Rafailov, Jeff Schneider, Tengyang Xie, Stefano Ermon, Chelsea Finn, and Aviral Kumar. Preference fine-tuning of LLMs should leverage suboptimal, on-policy data. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47441–47474. PMLR, 2024.

Yi-Shiuan Tung, Gyanig Kumar, Wei Jiang, Bradley Hayes, and Alessandro Roncone. CRED: Counterfactual reasoning and environment design for active preference learning. *arXiv preprint arXiv:2603.08531*, 2026. doi: 10.48550/arXiv.2603.08531. URL https://arxiv.org/abs/2603.08531.

Huaijie Wang, Shibo Hao, Hanze Dong, Shenao Zhang, Yilin Bao, Ziran Yang, and Yi Wu. Offline reinforcement learning for LLM multi-step reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 8881–8893. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.464. URL https://aclanthology.org/2025.findings-acl.464/.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024a. URL https://arxiv.org/abs/2406.04692.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024b.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023. URL https://arxiv.org/abs/2308.08155.

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 *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.757. URL https://aclanthology.org/2025.acl-long.757/.

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 *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76678–76692. PMLR, 2025a. URL https://proceedings.mlr.press/v267/zhang25cu.html.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025b.

Luhao Zhang, Xinyu Zhang, Linmei Hu, Dandan Song, and Liqiang Nie. Dually self-improved counterfactual data augmentation using large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5216–5227. Association for Computational Linguistics, 2025c. doi: 10.18653/v1/2025.acl-long.260. URL https://aclanthology.org/2025.acl-long.260/.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

A Additional Related Work

Hand-designed multi-agent scaffolds. Early LLM multi-agent systems largely relied on hand-designed interaction protocols. General frameworks such as AutoGen, MetaGPT, and CAMEL compose conversable or role-playing agents through prescribed communication interfaces and standardized operating procedures (Wu et al., 2023; Hong et al., 2024; Li et al., 2023). Multi-agent debate improves reasoning by repeatedly exposing models to one another's proposed answers (Du et al., 2024). AgentVerse and ChatDev organize specialized agents into prescribed collaboration stages or communication chains (Chen et al., 2024b; Qian et al., 2024). Self-improvement methods such as Self-Refine and Reflexion use iterative feedback or verbal reflection to revise a model's behavior at test time (Madaan et al., 2023; Shinn et al., 2023). These results, surveyed in Guo et al. (2024), established that structured interaction can improve over one-shot inference, but the coordination rule itself is primarily specified by humans. Later work instead optimizes the agentic system itself, as discussed in Section 2.

Model routing and heterogeneous worker selection. A complementary literature focuses on exploiting heterogeneity across LLMs. FrugalGPT studies cascades that trade off model quality and cost (Chen et al., 2023). RouteLLM trains routers from preference data to choose between a highly capable model and a more economical one (Ong et al., 2024). What's more, subsequent routing work estimates confidence or jointly optimizes model choice and test-time sampling budgets (Chuang et al., 2025; Ding et al., 2025). These methods generally select one model, or a cascade of models, for a query.

Multi-agent routing expands the decision space. MasRouter jointly chooses collaboration mode, agent roles, and LLM backbones (Yue et al., 2025), while Route-and-Reason decomposes a query into subtasks and trains a reinforced allocator to assign heterogeneous models according to estimated subtask complexity (Shao et al., 2025). Mixture-of-Agents instead aggregates multiple model outputs through a fixed layered construction (Wang et al., 2024a), and LLM-Blender ensembles candidate outputs by pairwise ranking and generative fusion (Jiang et al., 2023). Both improve over any single model by combining *outputs*, but the combination structure is fixed rather than a per-task decision the policy learns.

Our setting differs in two respects. First, the learned object is a *compositional workflow*: worker identity is only one decision alongside the natural-language instruction and information passed between calls. Second, our preference construction is designed to control routing shortcuts. Natural pairs are matched on final-worker identity and other coarse structural properties, while counterfactual interventions separately probe worker replacement, instruction edits, and access-structure edits. This distinction is important because surpassing the strongest fixed worker alone does not establish that a system has learned composition. Heterogeneous worker pools can provide substantial routing headroom. We therefore separately evaluate routing-only controls, oracle routing bounds, and task-level orchestration behavior. Our experiments additionally compare against an out-of-pool reference model, which no system in the paper is allowed to orchestrate.

Credit assignment for multi-step reasoning. The ambiguity of sequence-level workflow preferences is closely related to credit assignment in long-horizon reasoning. Outcome supervision provides a scalar label for an entire trajectory even when only a subset of its decisions is responsible for success or failure. Process reward models address this issue by scoring intermediate steps directly: Lightman et al. (2024) show that step-level human process supervision substantially outperforms outcome supervision for reasoning verification, while Math-Shepherd removes the human-annotation bottleneck by deriving step labels automatically from rollouts that reach a correct final answer (Wang et al., 2024b). OREO addresses credit assignment algorithmically with offline reinforcement learning, jointly learning a policy and value function from multi-step reasoning trajectories through a soft Bellman objective (Wang et al., 2025). Step-level Value Preference Optimization

(SVPO) instead uses tree search to obtain step-level comparisons and trains an explicit value model to complement preference optimization (Chen et al., 2024a).

These approaches seek training signals finer than sequence-level outcome supervision by estimating or annotating the value of intermediate reasoning states. Their steps, however, occur within one model's reasoning trace, whereas our steps are heterogeneous-worker calls whose value we probe by re-executing the workflow under a single-decision edit. Our approach is therefore complementary and operates primarily at the data level. Rather than learning a value function to infer where credit should go, we intervene on a structured orchestration action and observe whether the executed outcome changes. For a source workflow z, we construct an edited alternative $z' = \mathcal{I}_k(z)$ under an intervention $\mathcal{I}_k$ that changes one primary orchestration decision k, then freshly execute both workflows. When their observed outcomes differ, the resulting pair provides a comparison localized to that decision up to execution stochasticity. This converts an end-task verifier into a source of process-relevant supervision without requiring human step labels or a separately trained process reward model. After these counterfactual executions have been collected, policy optimization itself is offline.

Relation among the neighboring literatures. Prior work has made LLM systems increasingly compositional, adaptive, and learnable: routing methods exploit heterogeneous model strengths, graph and workflow methods optimize coordination structures, the Conductor learns free-form orchestration from end-to-end reinforcement learning, and offline reasoning methods address trajectory-level credit assignment. These threads can be separated along three axes: the *object* being learned (a fixed protocol, a graph or program, a single routing choice, or a free-form natural-language workflow), the *supervision signal* (hand design, structure search, terminal outcome, or step-level value estimates), and the *optimizer* (search, online RL, or offline preference learning). Our method targets a distinct combination of these axes: decision-localized, execution-grounded counterfactual preferences over a free-form orchestration workflow, optimized offline.

B WORKER POOL COMPOSITION

The orchestrated pool is a fixed set of seven heterogeneous LLM workers and is held constant across all orchestration systems. Table 4 reports their standalone performance on the 198-task evaluation set, using worker temperature 0.2 and three worker-sampling draws. No single worker dominates all four domains. Each shows a different profile of strengths, which gives orchestration complementary capabilities to combine. All workers use a 4096-token completion budget.

Table 4: The fixed seven-worker pool and standalone accuracy on the evaluation set. Macro is the unweighted mean of the four domain accuracies. Oracle rows are non-learnable diagnostics; `fable5` is an out-of-pool reference and is never orchestrated by CPO-ORCH.

Worker	Role	math	mmlu	rlpr	LCB	Macro
`sonnet46`	frontier model	79.4	97.2	72.8	83.3	83.18
`gpt55`	frontier model	75.6	97.2	71.1	85.2	82.27
`gpt-oss-120b`	strong open model	76.1	92.8	71.7	68.5	77.27
`haiku45`	small-fast frontier model	75.6	93.9	67.8	68.5	76.44
`deepseek-v3.2`	reasoning model	69.4	96.1	65.0	72.2	75.69
`qwen3-coder`	code specialist	78.3	87.2	54.5	66.7	71.67
`gemma3-27b`	small open model	79.5	81.7	37.2	51.9	62.55
Oracle per-domain router	best worker per domain	78.3	98.3	73.3	85.2	83.77
Oracle best-of-pool	any-worker-correct	90.0	98.3	86.7	90.7	91.44
`fable5`[‡]	out-of-pool reference	81.1	90.0	75.6	85.2	82.97

[‡] Not orchestrated by any system in this paper.

C ROBUSTNESS TO ORCHESTRATOR DECODING TEMPERATURE

Our main experiments decode the orchestrator greedily (temperature 0). To check that the learned behavior does not rely on argmax decoding, we re-evaluate CPO-ORCH and Natural DPO with orchestrator temperature 1.0, together with matched greedy controls, in a separate evaluation campaign. Sampling lowers macro accuracy by 2.97 points for CPO-ORCH and by 0.65 points for Natural DPO. The learned workflow structure, however, is essentially unchanged: mean CPO-ORCH depth moves from 2.202 to 2.200, and total worker calls change by less than 3%. Sampling therefore mainly perturbs individual choices within a workflow rather than the overall shape of the workflows the policy produces. It also increases variance: the across-seed standard deviation of CPO-ORCH rises from 0.08 to 1.04 points. Since the policy is trained to concentrate probability on preferred workflows, greedy decoding is the natural way to use it, and it removes an avoidable source of orchestrator-sampling variance. Therefore, we use it as our primary protocol. Because this campaign was run separately, its absolute accuracies are not directly comparable to Table 1.

D EVALUATION PROTOCOL AND SOURCES OF VARIANCE

All reported macro accuracies average the four equally weighted domains on the same 198-task evaluation set (mathematics 60, MMLU 60, RLPR 60, LiveCodeBench 18). For the main comparison, each trained system is evaluated at a predetermined final checkpoint over three worker-sampling draws, and these draws measure execution noise rather than training-seed variation. The mixing sweep and operator-family ablation use the stronger design of three training seeds per arm and three worker-sampling draws per checkpoint.

Under greedy orchestrator decoding, the emitted workflow is deterministic for a fixed checkpoint, so the variation across evaluation draws comes from workers sampled at temperature 0.2. Re-executing the same checkpoint at the same evaluation seed with an independent worker draw agrees on 91.41% of tasks: 17 of 198 outcomes flip, with 11 of the flips in RLPR.

E ADDITIONAL ANALYSIS OF THE COUNTERFACTUAL MIXING SWEEP

The main text fixes the number of preference pairs at 2034 and varies the counterfactual share ρ. Pair count is therefore controlled by construction, but the number and composition of source tasks are not. The seven arms cover $532, 578, 585, 576, 559, 513$, and 352 unique tasks for $\rho = 0, 0.25, 0.40, 0.50, 0.60, 0.75$, and 1.0, respectively. Task coverage therefore varies by a factor of 1.66 without ordering model performance: the widest-coverage arm ($\rho = 0.40$, 585 tasks) reaches 83.19%, while the narrowest ($\rho = 1.0$, 352 tasks) reaches 81.91%. The counterfactual-only endpoint uses the same number of rows as the natural-only endpoint while covering 180 fewer tasks.

The underlying pools also reach different parts of the training distribution. The natural pool contains 2602 pairs over 662 tasks, whereas the counterfactual pool contains 2502 pairs over 429 tasks. Of those 429 counterfactual tasks, 110 have no decisive natural pair and contribute 644 counterfactual pairs. Most of these counterfactual-only tasks are in RLPR: 97 of the 110 tasks and 621 of the 644 pairs. At the pool level, natural versus counterfactual unique-task coverage is $203/100$ for math, $191/21$ for MMLU, $52/122$ for RLPR, and $216/186$ for LiveCodeBench.

Because the two pools cover different parts of the training distribution, changing ρ also shifts domain composition: across the sweep, the MMLU share of exposures moves from 28.0% to 3.0% and the RLPR share from 7.8% to 31.0%. Pair structure also differs: counterfactual pairs preserve workflow depth by construction, whereas 15.5% of natural pairs differ in depth and 26.9% carry the smaller preference margin. The sweep thus shows that combining the two sources is beneficial across these differing training compositions.

F PREFERENCE-PAIR SETS USED FOR TRAINING

Table 5 reports the measured composition of the preference sets used in the main comparison and ablations. “Exposures” counts optimizer examples, whereas “Distinct” counts unique chosen/rejected

plan pairs. These are identical for the two controlled ablations but not for the main-table preference-trained systems.

In the main comparison, CPO-ORCH and Natural DPO both use 2046 pair exposures, but CPO-ORCH draws on 1472 distinct pairs whereas Natural DPO draws on 816. Their replication factors are therefore 1.39 and 2.51, respectively. By contrast, every arm in the mixing sweep and operator-family ablation contains 2034 distinct pairs sampled without replacement.

A second difference is that some natural pairs contain identical serialized chosen and rejected plans. Such rows carry zero DPO gradient but still consume exposure budget. There are 217 such rows in the main Natural DPO set and 105 in the CPO-ORCH set. Across the mixing sweep, the count falls from 81 at $\rho = 0$ to 0 at $\rho = 1$. Thus the natural-only endpoint contains 1953 signal-bearing rows versus 2034 at the counterfactual-only endpoint. This does not change the observed flatness of the sweep, but it is another reason not to interpret the endpoint comparison as evidence that individual counterfactual pairs are intrinsically stronger.

Table 5: Measured composition of preference sets. "Exp." is pair exposures, "Distinct" is the number of unique chosen/rejected plan pairs, "Repl." is Exp./Distinct, "Ident." counts identical chosen/rejected plans, and "Tasks" is unique task coverage. The domain columns count exposures.

System / arm	ρ	Exp.	Distinct	Repl.	Ident.	Tasks	math	mmlu	rlpr	LCB
Systems in Table 1										
CPO-ORCH	0.50	2046	1472	1.39	105	445	1040	411	372	223
Natural DPO	0.00	2046	816	2.51	217	345	1032	663	220	131
Learned single-worker router	—	1492	1492	1.00	0	373	372	136	544	440
Mixing sweep; no arm samples with replacement										
ρ=0.00	0.00	2034	2034	1.00	81	532	638	570	158	668
ρ=0.25	0.25	2034	2034	1.00	64	578	610	440	266	718
ρ=0.40	0.40	2034	2034	1.00	54	585	590	366	353	725
ρ=0.50	0.50	2034	2034	1.00	48	576	587	315	399	733
ρ=0.60	0.60	2034	2034	1.00	38	559	569	264	450	751
ρ=0.75	0.75	2034	2034	1.00	22	513	567	179	523	765
ρ=1.00	1.00	2034	2034	1.00	0	352	538	62	631	803
Operator-family ablation										
Routing only	0.26	2034	2034	1.00	59	691	640	448	291	655
Non-routing only	0.26	2034	2034	1.00	59	697	640	448	291	655
Mixed	0.26	2034	2034	1.00	59	703	640	448	291	655

G COUNTERFACTUAL OPERATOR ABLATION DETAILS

The operator ablation compares three matched arms. Each contains the same 1500 natural pairs plus 534 distinct counterfactual pairs sampled without replacement and stratified to a common domain profile, so all three arms contain 2034 pairs and have counterfactual share $\rho \approx 0.26$. The arms differ only in which operator families provide the counterfactual component: non-final worker replacements ("routing"), instruction/subtask/access-edge edits ("non-routing"), or both. The mixed arm is a matched ablation arm and is not intended to reproduce the full CPO-ORCH training mixture.

Table 6: Matched counterfactual-operator ablation. Each arm is trained with three training seeds and each checkpoint is evaluated with three worker-sampling draws.

Counterfactual operators	Macro	math	mmlu	rlpr	LCB
Routing only	82.61	80.37	97.04	67.22	85.80
Non-routing only	82.92	80.56	96.48	69.45	85.18
Mixed	83.39	82.04	97.59	70.00	83.95

Both routing and non-routing edits provide useful counterfactual supervision, and the mixed arm is numerically strongest, suggesting that the two families may contribute complementary signals. The comparison has two scope limitations. First, the routing pool used here contains only *non-final* worker replacements, so it does not test replacing the worker that produces the final answer. Second,

the routing and non-routing counterfactual pools span different numbers of source tasks (190 and 363, respectively). Domain stratification removes the most direct composition difference but does not equalize this task coverage.

H PER-TASK ORCHESTRATION FORENSICS

The oracle results in the main text leave a 7.67-point gap between the per-domain router (83.77%) and oracle best-of-pool (91.44%), indicating substantial per-task complementarity. We inspect CPO-ORCH's executed workflows to understand how much of this opportunity it uses.

Across the 594 task–seed executions, 468 workflows contain two steps and 126 contain three. We observe that the orchestrator adds the third step mainly on harder tasks, where accuracy is lower, rather than using greater depth as a generic default. There remain 15 tasks on which some standalone pool worker succeeds while CPO-ORCH fails, so the learned policy does not extract all available complementarity. Conversely, CPO-ORCH solves two tasks (one mathematics and one RLPR) that no standalone pool worker solves in the corresponding standalone evaluation. Relative to the oracle per-domain router, CPO-ORCH gains seven tasks and loses five.

Where the remaining gap lies. CPO-ORCH remains 5.88 macro points below oracle best-of-pool, but this gap is unevenly distributed: it is 15.0 points in RLPR, 6.7 in mathematics, 1.8 in LiveCodeBench, and zero in MMLU, where CPO-ORCH already matches the oracle. The gap should also not be read entirely as learnable headroom. The oracle selects, after the fact and using evaluation labels, whichever worker happened to answer correctly. With seven stochastic workers, part of its advantage therefore reflects favorable sampling rather than a stable complementarity that any policy could exploit when planning a workflow in advance. This effect is largest where outcomes are least stable: the domain with the largest gap, RLPR, also accounts for most of the outcome flips under re-execution reported below. The remaining gap is therefore best understood as an upper bound that combines genuine per-task complementarity with execution noise. Closing the genuine part likely requires information unavailable to a policy that plans before execution, such as verifying intermediate outputs or selecting among multiple candidate answers.

To check sensitivity to worker sampling, we re-execute the same checkpoint at the same evaluation seed with an independent worker-temperature-0.2 draw. Outcomes agree on 91.41% of the 198 tasks, while only 17 tasks flip, 11 of them in RLPR, the domain using the free-form judge. This indicates that the observed task-level behavior is not dominated by a single favorable worker draw.

I TRAINING AND EVALUATION CONFIGURATION

Preference optimization (DPO). All preference-training runs use LoRA on the attention projections $\{q, k, v, o\}$, DPO coefficient $\beta = 0.1$, gradient accumulation 8, maximum sequence length 2560, a constant learning rate with paged 8-bit AdamW, and loss computed over the full serialized orchestration plan. The remaining optimization settings differ across systems and are listed in Table 7.

Each trained system in Table 1, including Natural DPO, was tuned separately, and we report the best configuration found for that system. Their recipes therefore differ: CPO-ORCH uses rank 32 for two epochs at learning rate 10^{-5}, whereas the best Natural DPO configuration uses rank 16 for one epoch at 5×10^{-6}. The main-table comparison is thus between systems at their respective best settings. In addition, every arm within the mixing sweep and operator-family ablation uses the same rank, learning rate, epoch count, and exposure budget. These fixed-recipe comparisons are used to analyze counterfactual supervision, rather than compare with main table.

All reported preference-trained systems are evaluated at their predetermined final checkpoint, with no checkpoint selection on the evaluation set. Unless stated otherwise, CPO-ORCH uses counterfactual fraction $\rho = 0.5$. Exact pair-set composition is reported in Table 5.

Online RL (GRPO). The GRPO extension warm-starts from the CPO-ORCH DPO checkpoint, while the cold-start control initializes from the preference-untrained policy. The two runs otherwise use the same training manifest, training seed, learning rate 10^{-6} with a cosine schedule and

Table 7: Preference-training configurations. Shared by all rows are DPO $\beta = 0.1$, LoRA on $\{q, k, v, o\}$, sequence length 2560, gradient accumulation 8, constant learning rate within a run, and paged 8-bit AdamW.

Arm	r/α	Trainable	Learning rate	Epochs	Steps
CPO-ORCH	32/64	30.7M	1×10^{-5}	2	510
Natural DPO	16/32	15.3M	5×10^{-6}	1	255
Mixing-sweep arms	16/32	15.3M	1×10^{-5}	2	508
Operator-family arms	16/32	15.3M	1×10^{-5}	2	508
Learned single-worker router	16/32	15.3M	1×10^{-5}	2	372

3% warmup, no KL penalty ($\beta = 0$), full-parameter updates, and 200 iterations. Each iteration uses six prompts with 16 sampled workflows per prompt, for 19,200 reward-bearing rollouts in total. Workers are re-executed online at every iteration. Each run takes just under 19 hours on one `p4d.24xlarge`. The orchestrator is sampled at temperature 1.0 during training and decoded greedily at evaluation, and workers use temperature 0.2.

Training cost of offline versus online optimization. The two training paradigms differ sharply in where their cost is incurred. CPO-ORCH executes workers only while constructing its preference data. DPO optimization then runs on this fixed, pre-collected set for 510 LoRA steps, takes about 1.5 hours on three GPUs (roughly 4.5 GPU-hours), and issues no worker calls at all. The data-construction cost is paid once and is reusable: every arm of the mixing sweep and the operator-family ablation, 30 training runs in total, draws on the same natural and counterfactual pools without any additional worker execution. By contrast, GRPO must execute workers online at every iteration, requiring 19,200 reward-bearing rollouts and just under 19 hours on an eight-GPU `p4d.24xlarge` node (roughly 150 GPU-hours on the same GPU type, over 30× the DPO stage) for each run, and this cost recurs for every new run, seed, or configuration. The longer training time of GRPO stems largely from its online rollouts: every iteration must execute the sampled workflows through the worker pool and wait for their rewards before the policy can be updated. In our experiments, the online stage adds this cost without a clear accuracy gain over the offline checkpoint (Sec. 4.4), which makes offline preference optimization the more practical choice in our setting.

J INFERENCE COST DETAILS

Table 8 extends the main call-count comparison to every system for which execution telemetry was recorded. Cost comparisons are computed from measured execution traces rather than reconstructed from a price list. Because absolute API prices change over time and depend on provider pricing and caching, we report only relative cost differences. Where a cost trace comes from an earlier evaluation campaign, we use only its measured inference cost and report macro accuracy from the current evaluation in Table 1.

Table 8: Worker calls per task on the evaluation set, with macro accuracy taken from Table 1.

System	Calls/task	Macro
Fixed worker (`gpt55`)	1.00	82.27
Train-domain router (heuristic)	1.00	82.92
Best fixed multi-agent scaffold	2.00	84.02
CPO-ORCH → GRPO (warm)	2.197	84.58
CPO-ORCH	2.21	**85.56**
Compute-matched random workflow	2.22	67.66
Conductor (GRPO-only, cold)	2.409	79.25
Learned single-worker router	2.62	75.47

At a similar worker-call budget, CPO-ORCH incurs approximately 14% lower measured API cost than the best fixed multi-agent scaffold, while also achieving higher macro accuracy (85.56% versus 84.02%). The compute-matched random workflow costs about 56% less than CPO-ORCH but

reaches only 67.66% macro accuracy. Thus the improvement from CPO-ORCH is not explained simply by issuing more calls or spending more on worker inference.

Warm-starting GRPO from CPO-ORCH reduces both worker-call volume and measured API cost relative to cold-start GRPO (2.197 versus 2.409 calls per task, and about 19% lower cost). The learned single-worker router costs about 18% more than CPO-ORCH despite being 10.09 macro points less accurate. Its action space selects a single worker at each decision, but the evaluation trajectory can still contain multiple worker calls because the orchestrator is queried again after a worker returns.

For CPO-ORCH itself, the mean is 2.212 worker calls per task, and it consumes an average of 1864 input tokens and 1257 output tokens per task. Call volume varies by domain: mathematics uses 2.32 calls per task, RLPR 2.25, and MMLU exactly 2.00.

K GENERATION BUDGETS

The orchestrator generates each plan under a 1024-token budget and each worker executes under a 4096-token budget at temperature 0.2. The orchestrator budget is not binding for the main systems: measured plan-parse coverage is 1.000 for CPO-ORCH, 0.9949 for Natural DPO, and 1.000 for warm-start GRPO. Cold-start GRPO has parse coverage 0.9798, and the learned single-worker router 0.9495. For the latter, the missing cases are the 10 of 198 tasks on which it emits no usable step.

The worker budget does bind in a small number of standalone coding evaluations. In a LiveCodeBench audit of our API-only evaluation runs, 12 of 594 executions (about 2%) return empty text after reaching the 4096-token limit and are scored incorrect. All 12 come from single-model runs, and 8 of them occur on a single task. They arise when a model's extended reasoning is not fully accommodated by the shared token budget. We keep the common 4096-token budget unchanged so that standalone and orchestrated evaluations use the same worker configuration.

L PROMPTS AND FROZEN TEMPLATES

The orchestration interface is fully specified by four frozen artefacts: the orchestrator prompt template, the worker descriptions substituted into it, the in-context examples, and the templates the instruction-edit counterfactual operators draw from. We reproduce the prompt template, the worker descriptions, the counterfactual edit templates, and the two reconstructed in-context examples here. The two examples inherited verbatim from Nielsen et al. (2026) are referenced rather than reprinted. The first three artefacts are held fixed across every *learned* system we compare — both preference-trained policies, the learned single-worker router and both online-RL runs — and are hashed as a unit: a SHA-256 freeze manifest is computed over the concatenation of the template, the rendered example block, the rendered worker block and the parser version, so any drift in the interface invalidates the run rather than silently changing it. The fixed workers and fixed scaffolds have no orchestrator and therefore never see any of it.

Orchestrator prompt template. The template is a reproduction of the prompt introduced with the prior orchestration formulation (Nielsen et al., 2026), not a contribution of this work. We reproduce it in full because it defines the action space every system in the paper writes into. Three fields are substituted at render time: {`few_shot_examples`}, {`user_question`} and {`available_models`}. Two conventions in the last paragraph are worth noting, because the counterfactual access-edge operator acts on exactly them: the string `"all"` in an access list grants a step every previous step's subtask and response, and an empty list grants it none.

```
Your role as an assistant involves obtaining answers to questions by an
iterative process of querying powerful language models, each with a different
skillset.

You are given a user-provided question and a list of available numbered language
models with their metadata. Your objective is to output a sequence of up to 5
workflow steps.

Each routing is made of three elements: A language model, its assigned subtask
```

```
to accomplish, and an "access list" of past workflow steps it will see in its
context when trying to accomplish the subtask.

A subtask could directly ask the language model to solve the given question from
scratch, refine the solution of the previous subtask in the sequence, or perform
any other completely different task that would facilitate later language models
in the sequence to answer the original question with their expertise.

Based on your answer, the first model selected will be prompted with the user
question and the first subtask you define. Each following model in the sequence
will be prompted with the history of the previous subtask and response messages
specified in its access list, and will be asked to accomplish its relative
subtask. The answer of the final model and subtask will be provided back as the
final solution to the user.

Your response should be provided as three Python lists.

The first list should be called model_id, and contain the integers corresponding
to the numbered language models in the sequence you want to prompt.

The second list should be called subtasks, and contain the strings that will be
used to prompt the corresponding language model specified in model_id.

The third list should be called access_list, and contain the lists of past
routing messages (subtasks and assistant responses) from the previous routing
steps to include in the context in the current routing step.

You can pass the string "all" for any of the routing steps in access_list to
provide all the previous routing messages in the language model's context.
Alternatively, if you want an agent to attempt its subtask without any access to
previous routing steps, you can pass an empty list.

For instance:

{few_shot_examples}

USER QUESTION:
{user_question}

AVAILABLE LANGUAGE MODELS:
{available_models}
```

Worker descriptions. The seven strings below are substituted for {available_models}, rendered one per line as Model k: ⟨description⟩. Workers are identified to the orchestrator by *ordinal only*: no vendor or model name ever appears in the prompt, so the policy cannot condition on brand and the worker identities in Table 4 are not recoverable from the context. The descriptions are ours and are held fixed across all learned systems.

```
Model 0: High-capability general reasoning, mathematics, coding, and careful
verification; reliable strict output formatting.
Model 1: Strong general reasoning, mathematics, and coding; reliable strict
output formatting.
Model 2: Open-weight large model; solid general reasoning and mathematics with
concise answers.
Model 3: Reasoning-oriented model; strong on multi-step quantitative problems
and derivations.
Model 4: Code-and-structure specialist; precise symbolic manipulation and
formatted output.
Model 5: Smaller open model; weaker overall but solves some problems the
frontier models miss (highest per-task disagreement).
Model 6: Small fast model; concise direct answers, different capacity point from
the frontier models.
```

In-context examples. Four examples are substituted for {few_shot_examples}, each giving a question, a short rationale and the three lists. Two are inherited verbatim from the prior formulation's own prompt figure (Nielsen et al., 2026) and we do not reprint them here. They cover a biomedical multiple-choice question answered by a three-step workflow and a limit evaluation answered by a four-step one. The prior work lists two further arithmetic-search examples without printing them, so we reconstructed those two in the same format. Both reconstructions are given below. Across the four examples, workflow depth ranges from two to four steps and both access conventions appear, which is the only shape information the prompt supplies.

```
EXAMPLE 1
Question: Using the numbers 3, 7, 8, and 10 each at most once and the operations
```

```
+ - * /, make an expression equal to 24. Provide the final expression in
<answer> </answer> tags.
Assistant Response: This is a Countdown-style arithmetic search. A single strong
reasoner can search the small space, and a second model can verify the
arithmetic before we return it.
model_id = [4, 2]
subtasks = ['Find an arithmetic expression using the numbers 3, 7, 8, 10 (each
at most once) and + - * / that evaluates to 24. Show candidate expressions in
<idea> </idea> tags and give one working expression.', 'Verify that the proposed
expression evaluates exactly to 24 and uses each allowed number at most once,
then provide the final expression in <answer> </answer> tags.']
access_list = [[], ['all']]

EXAMPLE 2
Question: Using the numbers 2, 5, 6, 9 each at most once and the operations + -
* /, make an expression equal to 17. Provide the final expression in <answer>
</answer> tags.
Assistant Response: A short, self-contained search problem. One model proposes,
one model checks.
model_id = [0, 3]
subtasks = ['Find an arithmetic expression using the numbers 2, 5, 6, 9 (each at
most once) and + - * / that evaluates to 17. Explore options in <idea> </idea>
tags.', 'Check the proposed expression for arithmetic correctness and the
each-number-at-most-once rule, then return the final expression in <answer>
</answer> tags.']
access_list = [[], ['all']]
```

Counterfactual edit templates. Two of the five counterfactual operators rewrite instruction text, and both draw from a frozen family rather than from a model, so the edit is reproducible from the source workflow alone and no generator is in the loop (Sec. 3.3). The final-synthesis-instruction operator replaces the last step's subtask with one of the five directives below. The single-subtask operator *prepends* one of the four prefixes to a non-final subtask, leaving its text otherwise intact so the edit is intent-preserving and bounded in edit distance. In both cases the family member is selected by a deterministic index derived from the source, so one source workflow always yields the same edit.

```
Final-synthesis-instruction edit -- replaces the final subtask:
  (1) Synthesize the previous results without recomputing, and state the final
answer inside <answer></answer> tags.
  (2) Preserve the exact numeric result from the previous step and output only
the final answer inside <answer></answer> tags.
  (3) If the previous steps disagree, explicitly resolve the disagreement, then
give the final answer inside <answer></answer> tags.
  (4) Output only the required answer in the exact requested format inside
<answer></answer> tags.
  (5) Verify the candidate answers from the previous steps before selecting,
then give the final answer inside <answer></answer> tags.

Single-subtask edit -- prepended to a non-final subtask:
  (1) "Show all intermediate steps. "
  (2) "Reason carefully and double-check each step. "
  (3) "Work through this methodically. "
  (4) "Be explicit about assumptions. "
```

M QUALITATIVE EXAMPLES

This section gives concrete instances of the two objects the paper works with: a counterfactual preference pair used for training, and workflows emitted by the trained CPO-ORCH orchestrator on evaluation tasks. Worker indices follow the pool order used in the orchestrator prompt (Appendix L): 0 `sonnet46`, 1 `gpt55`, 2 `gpt-oss-120b`, 3 `deepseek-v3.2`, 4 `qwen3-coder`, 5 `gemma3-27b`, 6 `haiku45`. Subtask strings are abbreviated with "…" where long, and access lists are shown only where they were recorded in full.

A counterfactual preference pair. The pair below was produced by the final-worker replacement operator (A) on a mathematics task asking for the area of a triangle $\triangle OAB$. The two workflows have identical subtasks and access lists and differ only in the worker assigned to the final step. Both were re-executed, and exactly one succeeded, so the pair is retained as a decisive, decision-localized preference (Sec. 3.3).

```
rejected (R = 0):  model_id = [deepseek-v3.2 -> sonnet46]
chosen   (R = 1):  model_id = [deepseek-v3.2 -> gpt55]
subtasks, access_list: identical in both workflows
```

Scaffolding: same final worker, different outcome. On the mathematics task "find the domain of $f(x) = (2x-7)/\sqrt{x^2-5x+6}$", `sonnet46` answering alone is incorrect. CPO-ORCH keeps `sonnet46` as the final worker but precedes it with an analysis step by `deepseek-v3.2`. With that draft in context, the final answer ($x < 2$ or $x > 3$) is correct. The gain comes from the preceding step rather than from a change of final worker.

```
model_id    = [3, 0]        # deepseek-v3.2 -> sonnet46
subtasks    = ['Analyze the conditions for the square root ...',
               'Verify the analysis and refine the final answer ...']
                                                              -> correct
```

Routing: a different final worker. On an MMLU moral-scenarios question, `sonnet46` answering alone is incorrect. CPO-ORCH uses `sonnet46` to analyze both scenarios and routes the final multiple-choice decision to `gpt55`, which answers correctly. Here the learned policy selects a final worker other than the strongest standalone one, consistent with the task-dependent routing reported in Appendix H.

```
model_id    = [0, 1]        # sonnet46 -> gpt55
subtasks    = ['Analyze both scenarios ...',
               <final multiple-choice answer>]
                                                              -> correct
```

Composition: a task no single pool worker solves. The following mathematics task is one of the tasks, noted in Appendix H, that no worker solves in the standalone evaluation: "Let $ABCD$ be a regular tetrahedron with side length 2. The plane parallel to edges AB and CD and lying halfway between them cuts $ABCD$ into two pieces. Find the surface area of one of these pieces." CPO-ORCH decomposes it into three roles: `deepseek-v3.2` analyzes the geometry of the cross-section (a unit square), `sonnet46` computes the areas of the original and newly created faces, and `qwen3-coder` verifies the calculation and formats the result. The workflow returns the correct answer, $1 + 2\sqrt{3}$.

```
model_id    = [3, 0, 4]     # deepseek-v3.2 -> sonnet46 -> qwen3-coder
subtasks    = [
  'Analyze the geometry of a regular tetrahedron with side length 2.
   Determine the shape of the cross-section formed by a plane parallel
   to edges AB and CD and lying halfway between them...',
  'Calculate the surface area of one of the two pieces... Include the
   contribution from the original faces and the newly created
   cross-sectional face.',
  'Verify the calculations, and present the final surface area in exact
   mathematical form with proper formatting.'
]
access_list = [[], ['all'], ['all']]                          -> correct
```