CoRefine: Joint Reconfiguration of Concurrent Agent Workflows

Anonymous authors
Paper under double-blind review

Abstract

Agentic workflows sharing limited inference capacity must balance the value of additional sampling and repair against the delays imposed on concurrent tasks. We propose CoRefine, a runtime orchestration system that jointly selects pending workflow configurations within a sample-and-repair grammar. The controller adjusts sampling budgets, repair operations, and model assignments while preserving completed outputs and already-dispatched calls. It shortlists configurations using predicted success probabilities and execution costs, then ranks configuration pairs with a lightweight contention heuristic. In 1,024 BigCodeBench episode settings, CoRefine reduces token usage by 68.8% and mean aggregate flow time by 35.3% relative to a fixed-workflow baseline, with observed success rates of 81.0% and 80.0%, respectively. These results demonstrate the potential of runtime structural adaptation to improve concurrent inference efficiency, although retaining more computation can be preferable under light load and greater serving capacity.

1 Introduction

LLM-based multi-agent workflows solve complex tasks by coordinating model calls and tool operations (Li et al., 2023; Wu et al., 2023; Hong et al., 2023). Orchestration research improves model choice and workflow design (Ong et al., 2025; Zhuge et al., 2024; Zhang et al., 2025). When workflows share inference capacity, additional sampling or repair in one can delay another. After partial execution, completed outputs provide evidence about whether that additional computation remains worthwhile.

Existing approaches largely either adapt an individual workflow's structure (Niu et al., 2025) or coordinate concurrent workflows whose structures remain fixed (Lin et al., 2024; Luo et al., 2026; Dai et al., 2025). They do not jointly determine how much pending computation each workflow should retain when the workflows compete for shared inference capacity.

We introduce CoRefine to make this decision at a new task's arrival. The central idea is to reconfigure only pending computation: execution evidence estimates its value, and serving conditions estimate its interference with concurrent workflows. The controller jointly chooses the in-flight task's remaining configuration and the arriving task's complete configuration, adapting sampling, repair, and model assignments within a predefined sample-and-repair grammar. Completed outputs and dispatched calls remain unchanged.

Across 1,024 BigCodeBench episode settings spanning task difficulties, background loads, and serving capacities, CoRefine reduces tokens by 68.8% and mean aggregate flow time by 35.3% relative to Fixed, with observed success rates of 81.0% and 80.0%, respectively.

The main contributions of this paper are as follows:

1. **Problem formulation.** We formulate arrival-triggered reconfiguration of pending computation in two overlapping workflows, with shared inference capacity coupling their completion times.
2. **Method.** We combine profiler-guided shortlisting with contention-aware joint selection to search pending sampling, repair, and model assignments without exhaustively scoring every configuration pair online.

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.

3. **Runtime system.** We integrate profiling and configuration updates with workflow execution, preserving completed outputs and dispatched calls without additional online model inference.
4. **Empirical evaluation.** We evaluate concurrent coding workflows across task difficulties, background loads, and serving capacities, identifying both efficiency gains and conditions where retaining computation or independent adaptation is competitive.

The design comprises a Workflow Profiler, a Local Candidate Selector, and a Contention-Aware Joint Reranker (Section 4).

2 Related Work

Multi-agent LLM orchestration. Research on agentic orchestration optimizes how LLM calls are organized to solve complex tasks. Relevant work spans three directions: designing individual workflows, coordinating concurrent workflows, and adapting workflow structures during execution.

(i) Single-task model selection and workflow design. Model routing adapts the model assigned to a computation without changing the workflow (Ong et al., 2025). Other methods optimize workflow structure before deployment. GPTSwarm learns node prompts and graph connections (Zhuge et al., 2024), while ADAS and AFlow search over code-represented agentic systems (Hu et al., 2025; Zhang et al., 2025).

(ii) Multi-task optimization with fixed workflows. Workflow-aware serving coordinates calls from concurrent applications. Parrot exposes data dependencies through semantic variables to enable application-level serving optimizations (Lin et al., 2024). Agentix uses program execution history to prioritize and preempt LLM calls (Luo et al., 2026). Aragog adapts model assignments at each workflow stage according to current serving conditions, selecting among accuracy-preserving configurations without changing the workflow graph (Dai et al., 2025).

(iii) Single-task optimization with dynamic workflows. Flow represents workflows as activity-on-vertex graphs and updates their nodes and dependencies based on intermediate outputs and execution errors (Niu et al., 2025). DyFlow uses a designer–executor architecture to generate and revise subgoals using intermediate feedback (Wang et al., 2025). ADaPT recursively decomposes tasks when the executor cannot complete them, adapting the execution plan to task complexity and model capability (Prasad et al., 2024).

These directions motivate combining structural adaptation with cross-task coordination. CoRefine jointly reconfigures concurrent workflows using execution evidence and shared resource conditions, rather than treating their configurations as independent decisions.

LLM routing and cost-aware inference. Query-level routing and cascades optimize the quality–cost trade-off of individual requests. FrugalGPT learns conditional model cascades (Chen et al., 2024a), while RouteLLM selects between stronger and weaker models (Ong et al., 2025). These approaches decide which model to invoke or whether to make another call for a given query. CoRefine extends this quality–cost perspective to joint workflow reconfiguration, evaluating the pending configurations of concurrent workflows under shared serving constraints.

Across both lines of work, a workflow's configuration is settled when it starts, and later arrivals are admitted by the serving layer without revisiting the workflows already running. CoRefine instead makes a mid-execution arrival the decision point: it reconfigures the in-flight workflow's pending computation together with the arriving workflow's plan.

3 Problem Formulation

An agentic workflow is a computation graph $G = (V, E)$ whose nodes V perform model calls or tool operations and whose directed edges E specify data dependencies and control flow (Zhang et al., 2025). An LLM node $N_j = (M_j, P_j, \eta_j, \theta_j, n_j, O_j)$ is a logical computation unit, such as a writer or debugger, configured by:

- **Model M_j:** The language model invoked by the node.

- **Prompt** P_j**:** The role-specific instruction template, instantiated with the task and preceding outputs.
- **Inference budget** η_j**:** The budget for one sample, specified by a generation-token limit or a reasoning-effort setting when supported by the model.
- **Temperature** θ_j**:** The decoding parameter controlling output randomness.
- **Sampling count** n_j**:** The number of candidate outputs generated by repeated invocations of the node.
- **Output format** O_j**:** The output representation consumed downstream, such as executable code.

Edges. A directed edge $(N_u, N_v) \in E$ specifies the data dependency or control condition for executing node N_v.

Workflow configuration. We instantiate the graph model with a predefined sample-and-repair grammar. Structural adaptation changes sampling and repair operations; model reassignment changes their model tiers. A complete workflow plan $p = (w, n, \mathbf{m}, \mathbf{d}) \in \mathcal{P}$ is parameterized by:

- **Writer count** w**:** The number of writer slots.
- **Sampling budget** n**:** The number of samples generated per writer.
- **Writer models** $\mathbf{m} = (m_1, \ldots, m_w)$**:** The model assigned to each writer.
- **Repair sequence** $\mathbf{d} = (d_1, \ldots, d_r)$**:** The ordered model assignments for repair steps, with depth $r = |\mathbf{d}|$.

Observed state. For tasks arriving at $a_1 < a_2$, the controller makes one decision at $t = a_2$. Partition Task 1's operations into completed V_1^{done}, dispatched but unfinished V_1^{disp} (queued or running), and not-yet-dispatched V_1^{pend}. The state is

$$z_t = \big(\mathcal{H}_1,\ v_1,\ V_1^{\text{disp}},\ V_1^{\text{pend}},\ \kappa_t\big), \tag{1}$$

where $\mathcal{H}_1$ records completed operations, outputs, and execution history; v_1 is visible-test evidence; and κ_t contains model-tier capacities, hardware type, and observed queue information or calibrated background waiting. Section 4.2 specifies which state features the profiler uses in our experiments.

Joint action space. A joint action is a configuration pair $x = (c, p) \in \mathcal{A}(z_t) = \mathcal{C}(z_t) \times \mathcal{P}$. It specifies the pending workflow configuration for each task: c is Task 1's feasible continuation, and p is Task 2's complete plan. A feasible continuation preserves the outputs of V_1^{done} and the call specifications of V_1^{disp}. Only pending operations may be added, removed, or reassigned within the sample-and-repair grammar; these continuations form $\mathcal{C}(z_t)$.

Optimization objective. For task $i \in \{1, 2\}$, Q_i is a task-dependent quality measure. In our coding experiments, task success is the binary outcome $Q_i \in \{0, 1\}$, equal to one only when the submitted program passes all hidden tests; the success rate is its mean across tasks. Flow time is $F_i = t_i^{\text{finish}} - a_i$, and T_i counts prompt and completion tokens. Each outcome is a function of the joint action $x = (c, p)$, the state z_t, and a random element ω that captures stochastic generation and serving conditions:

$$Q_i = Q_i(c, p, z_t, \omega), \qquad F_i = F_i(c, p, z_t, \omega), \qquad T_i = T_i(c, p, z_t, \omega). \tag{2}$$

Shared capacity couples flow times through queueing. Each task's quality and token use depend on its own configuration and generated outputs. Capacity therefore enters through serving dynamics, not hard resource-budget constraints. With time and token penalties $\alpha, \beta \geq 0$, the joint utility and optimization objective are

$$R_A(x, z_t, \omega) = Q_1 + Q_2 - \alpha(F_1 + F_2) - \beta(T_1 + T_2), \tag{3}$$

$$x^\star \in \arg\max_{x \in \mathcal{A}(z_t)} \mathbb{E}_\omega \big[R_A(x, z_t, \omega) \mid z_t\big]. \tag{4}$$

The objective trades expected aggregate quality against flow time and token cost; it provides no tail-latency, fairness, deadline, or priority guarantees.

4 CoRefine Design

4.1 Overview

At Task 2's arrival, CoRefine jointly selects a configuration pair through three components (Figure 1).

Workflow Profiler combines offline measurements with execution evidence to predict each configuration's success probability, remaining time, tokens, and model-tier demand.

Local Candidate Selector ranks each task's alternatives and retains a bounded shortlist with plan-preserving and minimum-cost anchors.

Contention-Aware Joint Reranker scores configuration pairs under shared serving capacity. The execution engine applies the selected configurations without changing completed outputs or dispatched calls.

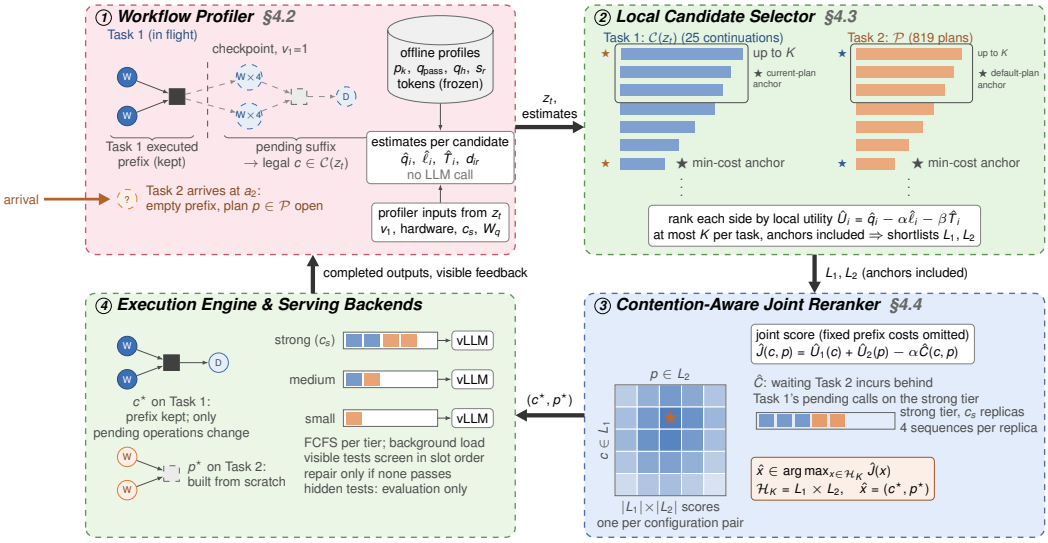

Figure 1: System overview of CoRefine.

4.2 Workflow Profiler

Success statistics and model-service costs are measured offline on development data. At arrival, the profiler combines these tables with visible-test evidence, hardware, replica count, and calibrated background wait. Estimates use lookup and aggregation, without additional model inference.

Execution state. Task 1 has generated one sample per writer and received visible-test feedback v_1, where $v_1 = 1$ indicates a passing candidate and $v_1 = 0$ indicates none. Its continuation $c = (b, h)$ specifies additional samples per retained writer and a repair sequence. Of the state z_t in Equation 1, the profiler uses v_1, c_s, the hardware type, and the calibrated background wait; hidden-test outcomes are unavailable.

Predicted success probabilities. Let q_{pass} be the development hidden-test success rate among visible-test passes, p_k the screening failure rate after k samples per writer, and q_h the success rate of repair sequence h on failed-screen cases. The profiler uses these statistics to predict Task 1's success probability:

$$\widehat{q}_1(c \mid v_1) = \begin{cases} q_{\text{pass}}, & v_1 = 1, \\ (1 - p_{1+b})q_{\text{pass}} + p_{1+b}q_h, & v_1 = 0. \end{cases} \tag{5}$$

Here p_{1+b} is a pooled proxy for continued screening failure. Without execution feedback, Task 2 uses a coarse plan-level prior based on its strongest writer tier and fixed bonuses for multiple sampling and repair (Appendix A).

Cost estimates. Profiled call costs yield remaining time $\widehat{\ell}_i$, tokens $\widehat{T}_i$, and service demand d_{ir} on tier r. Time includes visible testing and calibrated background waiting; tier demand sums service work even for overlapping calls. Task 1 weights repair costs by screening failure, whereas Task 2 charges configured repair rounds in full. Foreground interaction is added by the reranker. Appendix A gives the aggregation formulas.

4.3 Local Candidate Selector

Instead of jointly scoring all $|\mathcal{C}(z_t)|\,|\mathcal{P}|$ configuration pairs, the selector ranks each catalog by local utility:

$$\widehat{U}_i = \widehat{q}_i - \alpha\widehat{\ell}_i - \beta\widehat{T}_i. \tag{6}$$

Each shortlist retains at most K configurations. Task 1's current-plan anchor preserves its original pending continuation; Task 2's default-plan anchor preserves the canonical workflow. Each list also retains a minimum-cost anchor by replacing low-ranked entries when needed. The shortlists L_1, L_2 yield at most K^2 configuration pairs (49 for $K = 7$), allowing joint selection to trade local utility for lower shared-resource demand. Appendix A details anchor selection and catalog-ranking overhead.

4.4 Contention-Aware Joint Reranker

The reranker adds a heuristic penalty for foreground contention at the shared strong-model backend to the local estimates.

Contention heuristic. Let $N_s^W(c)$ and $N_s^D(h)$ count Task 1's remaining strong-model generation and repair calls, with profiled per-call service times s_s^W, s_s^D. Its outstanding work and the resulting contention-delay estimate are

$$\begin{aligned}\widehat{D}_{1s}(c) &= N_s^W(c)s_s^W + (1-v_1)p_{1+b}N_s^D(h)s_s^D,\\ \widehat{C}(c,p) &= \mathbb{I}[d_{2s}(p) > 0]\,\widehat{D}_{1s}(c)/c_s.\end{aligned} \tag{7}$$

The repair term is failure-weighted; the charge applies only when Task 2 also needs the strong tier. Here c_s is the number of strong-model replicas. This service-work heuristic captures foreground coupling, not the exact serving queue; calibrated background waiting is already included in $\widehat{\ell}_i$.

Joint selection. The configuration-pair score is

$$\begin{aligned}\widehat{J}(c,p) = \widehat{q}_1(c) + \widehat{q}_2(p) - \alpha[\widehat{f}_1(c) + \widehat{\ell}_2(p) + \widehat{C}(c,p)]\\ - \beta[\widehat{T}_1(c) + \widehat{T}_2(p)],\end{aligned} \tag{8}$$

where $\widehat{f}_1$ includes Task 1's remaining time and fixed prefix-time charge. Incurred prefix costs are common to all candidates. The penalty $\alpha\widehat{C}$ couples the two choices; without interference, the objective separates (Appendix A.3).

The controller scores every configuration pair in $\mathcal{H}_K = L_1 \times L_2$ and chooses

$$\widehat{x} \in \arg\max_{x\in\mathcal{H}_K} \widehat{J}(x). \tag{9}$$

Only the selected configurations are dispatched. Visible tests govern output selection and early repair stopping. Our experiments use two foreground tasks and one arrival-triggered decision.

5 Experiments

5.1 Experimental Setup

Testbed. We serve Qwen2.5-Coder-3B-Instruct, Yi-Coder-9B-Chat, and Phi-4 as small, medium, and strong tiers using vLLM (Kwon et al., 2023). Strong endpoints run on L20 or RTX 5880 GPUs

with $c_s \in \{1, 2\}$ physical replicas, each supporting four concurrent sequences. Model and decoding settings appear in Appendix B.2.

Tasks. We use BigCodeBench (Zhuo et al., 2025), alternating test methods into visible tests for selection and repair and hidden tests for final scoring. Tasks are divided into Hard (H) and Easy (E) halves by the number of successful single-sample, no-repair visible-test probes across the three tiers (Appendix B.1).

Workflow configuration. The canonical plan uses strong and medium writers, five samples per writer, and one strong repair. Task 2 arrives after Task 1's first sample from each writer and visible screening, at a single evaluated checkpoint with $V_1^{\text{disp}} = \varnothing$ and observed v_1. The search covers 25 Task 1 continuations and 819 complete plans, varying sampling, repair, and model assignments. We use $K = 7$; the full configuration ranges are in Appendix A.1.

Baselines. We compare five controls under a shared evaluator, workflow grammar, and serving environment. Their capabilities are motivated by prior work on sample-and-repair workflows, model routing, and workflow adaptation (Chen et al., 2024b; Dai et al., 2025; Niu et al., 2025; Zhang et al., 2025); they are not reproductions of those systems.

- **Fixed**: retains both canonical sample-and-repair plans.
- **Model-only**: jointly reassigns models, keeping sampling and repair operations fixed.
- **Task-1-only**: adapts Task 1's continuation and keeps Task 2's default plan.
- **Task-2-only**: adapts Task 2's complete plan and keeps Task 1's original continuation.
- **Independent**: adapts both tasks through local selection without joint reranking.

CoRefine adapts both tasks using contention-aware joint selection.

Background traffic. Poisson workflow arrivals each submit five strong-model requests. The request-equivalent rate λ gives workflow rate $\lambda/5$ and normalized load $\rho = \lambda/\mu_{\max} \in \{0.20, 0.50, 0.90, 0.95\}$, where $\mu_{\max}$ is each hardware–capacity setting's measured saturated throughput. Variants share the same background arrival trace.

Metrics and protocol. We report hidden-test success rate, tokens, mean aggregate flow $F_1 + F_2$, per-task P95 flow, and utility R_A ($\alpha = 10^{-3}, \beta = 10^{-6}$). Flow includes queueing, visible testing, and blocking planning, but excludes hidden evaluation. Variants share prefix evidence and execution outcomes for identical configuration pairs, retaining method-specific planning costs. Paired-cluster 95% intervals quantify task-group uncertainty conditional on the reported run, not variability across generation seeds or runtime repetitions (Appendices C and B.2).

5.2 Background-Load Calibration

We calibrate each hardware–capacity–load setting for 240 seconds with two physical repetitions. Figure 2 shows increasing queue waits as arrival rates leave less spare capacity; five-request bursts also permit queueing below saturation. These measurements establish the background contention levels used in the workflow evaluation (Appendix B.3).

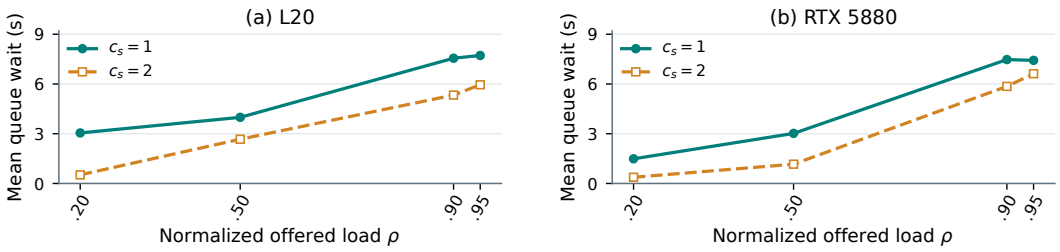

Figure 2: Mean queue wait in the physical λ sweep (two repetitions per setting).

5.3 Performance across Task Difficulties

Structural adaptation versus model routing. CoRefine attains the highest observed mean utility across the four task-pair categories, with token ratios of 0.22–0.39 and aggregate-flow ratios of 0.46–0.83 relative to Fixed (Table 1). Model-only retains all sampling and repair operations: its predicted strong-tier demand is 98.9 s, compared with 75.4 s for Fixed and 28.8 s for CoRefine, and its flow time exceeds Fixed despite token savings (Appendix D). Adapting pending operations reduces work submitted to the bottleneck rather than only reassigning that work.

Effect of workflow role. Task-1-only and Task-2-only each leave one workflow unchanged and realize only part of the cost reduction. The two roles also differ in available evidence: Task 1 can reuse screened outputs, whereas Task 2 must be configured from priors alone. Aggregate gains can conceal this asymmetry; in E→H, Task 2's observed success rate falls while total cost drops (Appendix D.2).

Table 1: Performance across task difficulties.

	H→E						H→H					
Method	Success rate (%)	Utility	Token ratio	Flow ratio	Aggregate flow (s)	P95 flow (s)	Success rate (%)	Utility	Token ratio	Flow ratio	Aggregate flow (s)	P95 flow (s)
Fixed	74.80	1.4232	1.000	1.000	55.2	53.8	69.73	1.3099	1.000	1.000	65.0	64.5
Model-only	74.80	1.4110	0.921	1.245	68.8	71.1	**72.27**	1.3512	0.935	1.166	75.7	77.7
Task-1-only	72.66	1.3852	0.738	0.994	54.9	74.7	68.55	1.2946	0.759	0.948	61.6	81.9
Task-2-only	75.59	1.4474	0.662	0.954	52.7	58.9	69.53	1.3238	0.629	0.839	54.5	56.8
CoRefine	**77.54**	**1.4981**	0.393	0.827	45.7	66.8	71.48	**1.3735**	0.388	0.747	48.5	67.2
	E→H						**E→E**					
Method	Success rate (%)	Utility	Token ratio	Flow ratio	Aggregate flow (s)	P95 flow (s)	Success rate (%)	Utility	Token ratio	Flow ratio	Aggregate flow (s)	P95 flow (s)
Fixed	**85.16**	1.6337	1.000	1.000	50.8	47.2	90.43	1.7445	1.000	1.000	47.3	45.8
Model-only	83.20	1.5811	0.944	1.288	65.4	68.4	**91.60**	1.7596	0.917	1.205	57.0	63.5
Task-1-only	84.57	1.6455	0.619	0.677	34.3	36.6	90.04	1.7634	0.575	0.587	27.7	28.5
Task-2-only	83.79	1.6182	0.624	0.906	46.0	45.4	90.23	1.7517	0.644	0.891	42.1	44.9
CoRefine	84.38	**1.6576**	0.240	0.502	25.5	29.9	90.43	**1.7834**	0.220	0.455	21.5	25.4

Note. 256 episode settings per task-pair category. Cost ratios use Fixed within each group. Aggregate flow is the mean of $F_1 + F_2$; P95 is computed from individual-task flow times. Bold marks the highest observed success or utility. Paired intervals: Appendix C.

Effect of prefix evidence. Every Easy-T1 checkpoint passes its visible screen. CoRefine then stops further Task 1 sampling and assigns Task 2 a smaller workflow; Independent selects the same configurations. Failed Hard-T1 prefixes retain the two-strong-repair suffix, but CoRefine selects two additional samples per writer where Independent selects four. The token ratio consequently falls from 0.53 to 0.39 in H→E and from 0.51 to 0.39 in H→H (Appendix D.1). Decision differences are concentrated in these unresolved prefixes; isolating the contention term requires the separate ablation described in Appendix C.

Success and tail-latency trade-offs. The aggregate success rates are close to Fixed, but the trade-offs differ by task-pair category. In E→H, CoRefine is 0.8 percentage points lower and its utility advantage comes from cost savings. In the Hard-T1 categories, P95 flow is 3–13 seconds higher despite lower mean flow, as unresolved prefixes retain sequential repairs. Paired uncertainty estimates and their interpretation remain in Appendix C.

5.4 Effect of Background Load

Token savings remain similar across loads (ratios round to 0.31), while CoRefine's flow ratio rises from 0.60 at $\rho = 0.20$ to 0.69 at $\rho = 0.95$. Absolute savings remain within 18–21 seconds (Table 2; Figure 3). The amount of foreground work removed changes little, but surviving calls face more background waiting. A similar absolute saving therefore becomes a smaller fraction of

total flow. Model-only retains greater strong-tier demand and incurs a growing flow penalty. These measurements describe the trend rather than an exact decomposition of queueing delay.

Table 2: Performance across background loads and serving capacities.

	$\rho = 0.20$					$\rho = 0.50$				
Method	Success rate (%)	Utility	Token ratio	Flow ratio	Aggregate flow (s)	Success rate (%)	Utility	Token ratio	Flow ratio	Aggregate flow (s)
Fixed	80.66	1.5494	1.000	1.000	45.7	80.47	1.5404	1.000	1.000	50.7
Model-only	**81.45**	1.5617	0.927	1.100	50.3	**80.66**	1.5374	0.932	1.161	58.9
Task-1-only	78.91	1.5292	0.675	0.801	36.6	78.71	1.5212	0.676	0.803	40.7
Task-2-only	80.47	1.5582	0.638	0.865	39.5	79.30	1.5302	0.639	0.870	44.1
CoRefine	81.25	**1.5920**	0.312	0.598	27.3	80.08	**1.5644**	0.312	0.620	31.5
	$\rho = 0.90$					$\rho = 0.95$				
Method	Success rate (%)	Utility	Token ratio	Flow ratio	Aggregate flow (s)	Success rate (%)	Utility	Token ratio	Flow ratio	Aggregate flow (s)
Fixed	79.49	1.5102	1.000	1.000	61.5	79.49	1.5114	1.000	1.000	60.3
Model-only	79.69	1.4977	0.931	1.287	79.1	80.08	1.5061	0.930	1.302	78.5
Task-1-only	79.10	1.5194	0.677	0.819	50.4	79.10	1.5189	0.675	0.844	50.9
Task-2-only	79.49	1.5230	0.640	0.898	55.3	79.88	1.5297	0.640	0.935	56.3
CoRefine	**81.84**	**1.5900**	0.312	0.667	41.0	**80.66**	**1.5662**	0.313	0.687	41.4
	$c_s = 1$					$c_s = 2$				
Method	Success rate (%)	Utility	Token ratio	Flow ratio	Aggregate flow (s)	Success rate (%)	Utility	Token ratio	Flow ratio	Aggregate flow (s)
Fixed	79.98	1.5218	1.000	1.000	59.6	80.08	1.5338	1.000	1.000	49.5
Model-only	80.08	1.5076	0.931	1.292	77.0	**80.86**	1.5438	0.930	1.139	56.4
Task-1-only	79.20	1.5250	0.676	0.784	46.7	78.71	1.5193	0.676	0.860	42.6
Task-2-only	79.30	1.5215	0.640	0.887	52.8	80.27	1.5491	0.639	0.904	44.8
CoRefine	**81.15**	**1.5803**	0.312	0.622	37.1	80.76	**1.5760**	0.312	0.677	33.5

Note. Metrics follow Table 1. These are marginal views of the same 1,024 contexts: 256 per load, pooled over capacities, and 512 per capacity, pooled over loads. Paired intervals: Appendix C.

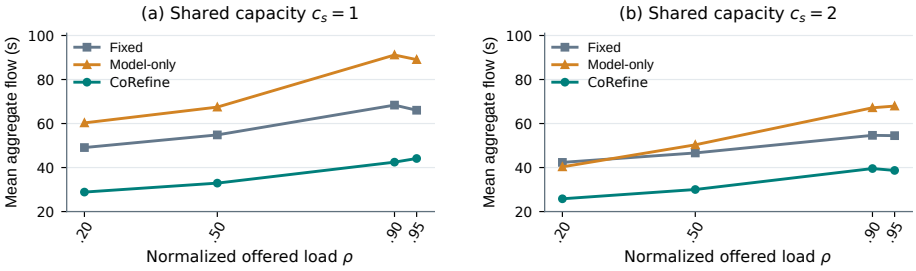

Figure 3: Mean aggregate flow under background load, with one or two strong-model replicas.

5.5 Effect of Serving Capacity

CoRefine's flow ratio is 0.62 with one strong-model replica and 0.68 with two, while its token ratio is 0.31 in both settings (Table 2). A second replica relieves competition for the strong tier, reducing the waiting that can be avoided by removing pending work. At two replicas and $\rho = 0.20$, Model-only instead has higher observed utility (1.5991 versus 1.5779): its 2.3-percentage-point success advantage outweighs CoRefine's cost savings (Appendix D.3). Retaining computation is therefore preferable in this lightly loaded, higher-capacity setting. Comparisons match normalized load, not absolute arrival rate.

5.6 Preference Sensitivity and Planning Overhead

Utility-coefficient sensitivity. We rescore fixed decisions and outcomes with $(\alpha, \beta) = \gamma(10^{-3}, 10^{-6})$ (Figure 4). CoRefine's utility advantage over the four controls shown increases with γ: H→E favors CoRefine for all $\gamma \geq 0$, and the other categories cross over by $\gamma = 0.50$. Cost savings grow in weight while success differences remain fixed, so reducing computation becomes more attractive as cost receives greater emphasis. This is fixed-outcome rescoring, not a rerun of the planner at new coefficients (Appendix D.4).

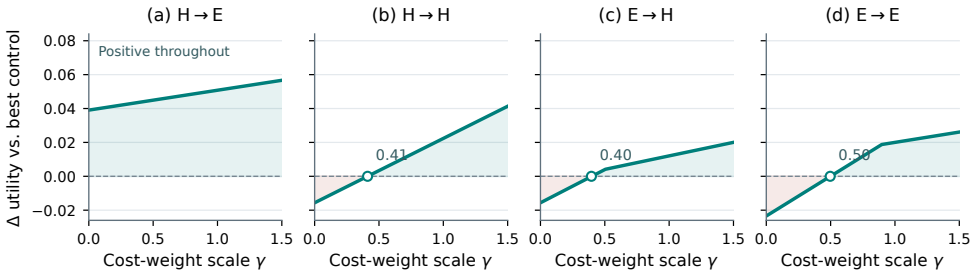

Figure 4: Fixed-outcome utility difference between CoRefine and the best of Fixed, Model-only, Task-1-only, and Task-2-only as cost weights scale by γ; $\gamma = 1$ is nominal.

Shortlist validation. In an earlier MBPP diagnostic with 18 measured plans per task, the $K = 7$ shortlist solver was within 0.01 of the exhaustive-oracle utility in 3,925 of 3,927 development cases across seven action scopes (99.95%). This result motivated $K = 7$; it supports near-optimality in that smaller space under the earlier objective, not exhaustive validation of the current 20,475-configuration-pair BigCodeBench search (Appendix A.4).

Online overhead. Planning averages 20.4 ms with profiles and catalogs preloaded. Profile lookup and bounded pairwise scoring avoid extra LLM inference.

6 Conclusion and Limitations

Our coding-workflow experiments show that adapting pending sampling and repair can reduce token use and mean aggregate flow. Independent adaptation selects the same configurations at successful Easy-T1 checkpoints; differences from joint selection occur at failed Hard-T1 checkpoints. The gains have limits: success rate or P95 flow can worsen, and Model-only has higher utility under light load with two replicas. The reported intervals condition on one generation and execution run, and a direct contention-term ablation remains outstanding. Richer success predictors, repeated arrivals, and larger sets of active workflows are the next steps.

AI use statement

Generative AI tools were used to draft portions of the manuscript text and to assist in writing and refactoring experimental code. All AI-generated text and code were reviewed, verified, and edited by the authors, who take full responsibility for the content of this paper.

Reproducibility statement

The separate experimental archive contains the planner, configurations, frozen decisions and profiles, and 6,144 execution records. This manuscript package includes the audited statistical export and scripts for regenerating uncertainty tables. Appendix B.2 describes execution and model settings; `REPRODUCIBILITY.md` records artifact checks and reproduction limits.

REFERENCES

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. URL https://web.stanford.edu/~boyd/cvxbook/.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=cSimKw5p6R.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/2460396f2d0d421885997dd1612ac56b-Paper-Conference.pdf.

Yinwei Dai, Zhuofu Chen, Anand Iyer, and Ravi Netravali. Aragog: Just-in-time model routing for scalable serving of agentic workflows. arXiv:2511.20975, 2025. URL https://arxiv.org/abs/2511.20975.

Sirui Hong, Mingchen Zhuge, Jiaqi 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. arXiv:2308.00352, 2023. URL https://arxiv.org/abs/2308.00352.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *ACM Symposium on Operating Systems Principles*, 2023. URL https://arxiv.org/abs/2309.06180.

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*, 2023. URL https://arxiv.org/abs/2303.17760.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of LLM-based applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 929–945. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/lin-chaofan.

Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. Agentix: An efficient serving engine for LLM agents as general programs. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 2443–2459. USENIX Association, 2026. URL https://www.usenix.org/conference/nsdi26/presentation/luo.

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*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ba84da6921f3040b74ee163aa7451f53-Paper-Conference.pdf.

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 from preference data. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5503a7c69d48a2f86fc00b3dc09de686-Abstract-Conference.html.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-needed decomposition and planning with language models.

In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4226–4252. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-naacl.264. URL https://aclanthology.org/2024.findings-naacl.264/.

Yanbo Wang, Zixiang Xu, Yue Huang, Xiangqi Wang, Zirui Song, Lang Gao, Chenxi Wang, Xiangru Tang, Yue Zhao, Arman Cohan, Xiangliang Zhang, and Xiuying Chen. DyFlow: Dynamic workflow framework for agentic reasoning. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2509.26062.

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:2308.08155, 2023. URL https://arxiv.org/abs/2308.08155.

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*, 2025. URL https://openreview.net/forum?id=z5uVAKwmjf.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=uTC9AFXIhg.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.15877.

A METHOD DETAILS

A.1 CONFIGURATION SPACE AND SHORTLIST CONSTRUCTION

Complete plans use $w \in \{1, 2, 3\}$ writers, $n \in \{1, 3, 5\}$ samples per writer, and $r \in \{0, 1, 2\}$ repair rounds. Writers use any of the three model tiers; repair uses medium or strong models. This gives 819 complete plans. Task 1 retains its writers and selects $b \in \{0, \dots, 4\}$ additional samples per writer and $h \in \{\varnothing, m, s, mm, ss\}$, giving 25 continuations and 20,475 configuration pairs. The canonical plan uses strong and medium writers, five samples per writer, and one strong repair.

Candidates are ranked by local score, then service demand and plan identifier. Each shortlist retains Task 1's original continuation or Task 2's canonical plan, plus the configuration minimizing $\widehat{\ell} + \widehat{T}/1000$. Anchors replace low-ranked entries, keeping each list at most $K = 7$ and its product at most 49 configuration pairs.

The implementation also searches a resource-price extension (Boyd & Vandenberghe, 2004), but the main experiment retains zero prices throughout and therefore uses local-utility ordering. With profiles and catalogs preloaded, planning averages 20.4 ms (median 21.0 ms; P95 23.7 ms), including catalog scans, price-vector evaluations, and joint scoring. This blocking time is charged to flow; no additional model inference is performed. The reduction from 20,475 to 49 pairs is not an end-to-end speedup factor.

A.2 SUCCESS-PROBABILITY AND COST ESTIMATES

Without Task 2 execution evidence, the profiler uses

$$\widehat{q}_2(p) = \tfrac{1}{2} \max_{j \le w} \tau(m_j) + 0.02\mathbb{I}[n > 1] + 0.02\mathbb{I}[r > 0], \tag{10}$$

where τ is 0.55, 0.70, or 0.85 for small, medium, or strong writers. The bonuses distinguish the presence of multiple samples or repair, not their counts; additional writers affect this prior only through the strongest tier.

Let s^W_m, s^D_m be profiled generation and repair times, $\bar{t}$ mean tokens per call, s^V visible-test time, and W_q the calibrated background wait. Task 2 costs are

$$\widehat{T}_2(p) = \bar{t}(wn + r), \tag{11}$$

$$\widehat{\ell}_2(p) = \max_j s^W_{m_j} + W_q\mathbb{I}[\exists j : m_j = s] + \sum_{j=1}^{r} \left(s^D_{d_j} + W_q\mathbb{I}[d_j = s]\right) + (1 + r)s^V. \tag{12}$$

This approximation treats generation as parallel, without increasing its latency with n, and charges all configured repair rounds even when execution stops early. Task 1 instead uses measured widening costs and failure-weighted repair costs. Tier demand sums service work under the same conventions; it is not GPU occupancy. Task 1's statistics pool development outcomes, with repair statistics shared across sampling budgets. Its passing-state estimate stays constant even when widening changes the selected visible-pass candidate.

These coarse profiles are used in the main experiments; Appendix E reports the separate post hoc calibration diagnostic. Online estimates use the visible-test result, hardware, replica count, and calibrated wait, never hidden-test outcomes.

A.3 COUPLING AND SHORTLIST LOSS

When independent selection suffices. Write the future objective as $U_1(c) + U_2(p) - \alpha C(c, p)$. An additive interference term $C(c, p) = C_1(c) + C_2(p)$ has zero mixed difference:

$$C(c, p) + C(c', p') - C(c, p') - C(c', p) = 0. \tag{13}$$

A nonzero mixed difference therefore rules out separability. With no mutual interference, selection reduces to independent local maximization, provided shortlisting retains the maximizers and scoring and tie-breaking agree. Low background load alone does not guarantee these conditions.

Shortlist loss versus estimation error. Let $x^\star$ maximize the true conditional objective J over $\mathcal{A}$, and $x^\star_{\mathcal{H}}$ maximize it over the retained set $\mathcal{H}$. Define $\Delta_{\mathcal{H}} = J(x^\star) - J(x^\star_{\mathcal{H}})$. If $|\widehat{J}(x) - J(x)| \le \epsilon$ on $\mathcal{H}$ and $\widehat{x}$ maximizes $\widehat{J}$ there, then

$$J(x^\star) - J(\widehat{x}) \le \Delta_{\mathcal{H}} + 2\epsilon. \tag{14}$$

The inequality follows from $J(x^\star_{\mathcal{H}}) \le \widehat{J}(x^\star_{\mathcal{H}}) + \epsilon \le \widehat{J}(\widehat{x}) + \epsilon \le J(\widehat{x}) + 2\epsilon$. It separates pruning loss from ranking error; the experiments do not establish a uniform bound on ϵ.

A.4 Small-space shortlist diagnostic

An earlier MBPP diagnostic compared 18 measured plans per task over 187 task pairs, three capacity/load settings, and seven action masks (3,927 cases). The shortlist solver was within 0.01 utility of the exhaustive within-action oracle in 3,925 cases (99.95%), with a recorded gap upper bound of approximately 0.018. This diagnostic selected $K = 7$ using latency/token weights $10^{-2}/10^{-5}$ and reconfiguration charges. It supports the earlier solver in that smaller space, not exhaustive validation of the current BigCodeBench objective or search space.

B Experimental Protocol

B.1 Task selection and pairing

We use 128 tasks from 644 sandbox-qualified tasks in the 1,140-task BigCodeBench benchmark. Each task receives one no-repair, single-sample visible-test probe from each model tier. Tasks are ordered by their number of passing probes, then task-identifier string; the lower and upper halves form 64 Hard and 64 Easy tasks. Hard has 22, 23, and 19 tasks with scores 0, 1, and 2; Easy has 20 and 44 with scores 2 and 3. These local strata can share probe scores and are not the official BigCodeBench-Hard subset.

Hard tasks in ascending order are matched with Easy tasks in descending order. The first 32 pairs form H→E; the remaining 32 reverse order to form E→H. H→H and E→E pair the corresponding task sides at matching indices. Each category has 32 disjoint task pairs, each evaluated at two capacities and four loads on its assigned hardware. This yields 1,024 episode settings and 6,144 method outcomes. Across categories, each task occurs in two pairs, forming 32 disjoint four-task components used for pooled bootstrap comparisons.

B.2 Models and execution

The small, medium, and strong checkpoints are `Qwen/Qwen2.5-Coder-3B-Instruct`, `01-ai/Yi-Coder-9B-Chat`, and `microsoft/phi-4`. Generation uses temperature 0.8 and a 4,096-token limit; servers use bfloat16 and an 8,192-token context limit. The runner uses `bigcode/bigcodebench`, split `v0.1.4`, combining instructions, code interfaces, and visible-test grounding. Scores therefore describe this adapted evaluation, not the unmodified leaderboard protocol. Per-call seeds depend on the run, episode, plan, writer slot, and sample index; different plans can use different sample streams.

Task 2 arrives after Task 1's first sample per writer and visible screening. All methods plan from the same completed outputs and pass bit. Each also replays the prefix for time/token accounting; replayed and shared pass bits differ in 453/6,144 records, but planning uses the shared evidence. Identical selected configuration pairs share one execution outcome while retaining method-specific blocking planning costs. The fixed workflow in either single-task control starts immediately, and selected configurations are checked against frozen decisions before dispatch.

B.3 Background-load calibration

Poisson background workflow arrivals submit five strong-model requests each. The request-equivalent rate is λ, and normalized load is $\rho = \lambda/\mu_{\max}$ for the relevant hardware and replica count. Each calibration lasts 240 seconds with two repetitions, giving 32 measurements. Tables 3 and 4 give throughput, request service times, and calibrated queue waits; equal normalized load need not mean equal absolute arrival rate.

Table 3: Reported saturated throughput and average request service time. Capacity denotes physical strong replicas; each permits four concurrent sequences.

Hardware	Capacity	$\mu_{\max}$ (requests/s)	Mean service (s)
L20	1	0.420	8.16
L20	2	0.941	8.05
RTX 5880	1	0.446	7.68
RTX 5880	2	0.987	7.71

Table 4: Offered request-equivalent rate λ, measured normalized offered load ρ, and mean queueing time W_q. Each background workflow contributes five requests.

Hardware	Capacity	Load	λ (req/s)	ρ	W_q (s)
L20	1	Low	0.0839	0.199	3.05
L20	1	Moderate	0.2098	0.497	3.99
L20	1	High	0.3776	0.894	7.55
L20	1	Near-sat.	0.3986	0.944	7.72
L20	2	Low	0.1882	0.199	0.52
L20	2	Moderate	0.4706	0.509	2.67
L20	2	High	0.8470	0.908	5.33
L20	2	Near-sat.	0.8941	0.952	5.95
RTX 5880	1	Low	0.0891	0.187	1.49
RTX 5880	1	Moderate	0.2228	0.491	3.02
RTX 5880	1	High	0.4010	0.912	7.47
RTX 5880	1	Near-sat.	0.4232	0.959	7.42
RTX 5880	2	Low	0.1974	0.200	0.37
RTX 5880	2	Moderate	0.4936	0.496	1.16
RTX 5880	2	High	0.8884	0.897	5.85
RTX 5880	2	Near-sat.	0.9378	0.950	6.62

C Statistical Uncertainty

C.1 Metrics and paired bootstrap

All differences are CoRefine minus the named baseline. Success rate is $100\ \mathrm{mean}[(Q_1 + Q_2)/2]$; utility, tokens, and aggregate flow average R_A, $T_1 + T_2$, and $F_1 + F_2$. Point-table cost ratios use aggregate Fixed costs. P95 compares the methods' quantiles of individual-task flow, not aggregate flow or quantiles of paired differences.

Within a task-pair category, we resample its 32 task pairs, retaining all eight capacity/load settings. For pooled categories, including load- and capacity-specific comparisons, we resample the 32 four-task components. Each draw uses the same clusters for every method and metric. We recompute means and P95 in 100,000 draws and report the 2.5th and 97.5th percentiles of their differences, using linear quantile interpolation at $(n-1)p$. The frozen reporting protocol specified Model-only comparisons in two load buckets and high-load Independent comparisons; the remaining analyses are exploratory. All intervals are marginal, without multiplicity adjustment.

These intervals measure task-group variation conditional on the observed profiles, decisions, outputs, and service times, not generation-seed or runtime-repeat variability. Each stratum has 32 clusters and at least 512 task flows, but a few clusters can dominate the upper tail, limiting P95 precision.

C.2 Results and interpretation

Tables 5 and 6 give overall and capacity-specific comparisons; Tables 7 and 8 separate task difficulty and background load. Token reductions are supported against all controls. Utility intervals include

zero against Model-only, Task-1-only, and Independent, as do mean flow against Independent and P95 against Fixed. Intervals crossing zero do not establish equivalence, and success intervals do not establish non-inferiority.

The joint selector with only its contention term removed was not run. Independent changes the selection procedure and is not a substitute for that ablation, which must hold catalogs, profiles, shortlists, anchors, and tie-breaking fixed.

Table 5: Overall paired differences and conditional uncertainty.

Baseline	Success rate (pp)	Utility	Tokens (10^3/task pair)	Mean aggregate flow (s)	P95 task flow (s)
Fixed	$+0.93$ $[-1.22, +3.37]$	$+0.0503$ $[+0.0071, +0.0985]$	-12.51 $[-13.28, -11.77]$	-19.3 $[-23.3, -15.8]$	-2.3 $[-12.4, +7.9]$
Model-only	$+0.49$ $[-2.15, +3.56]$	$+0.0524$ $[-0.0010, +0.1145]$	-11.24 $[-12.08, -10.46]$	-31.4 $[-34.4, -28.4]$	-19.0 $[-30.0, -7.2]$
Task-1-only	$+2.00$ $[-1.12, +5.42]$	$+0.0560$ $[-0.0073, +0.1255]$	-6.61 $[-7.04, -6.19]$	-9.3 $[-13.6, -6.3]$	-14.8 $[-26.5, -4.7]$
Task-2-only	$+1.17$ $[0.00, +2.98]$	$+0.0429$ $[+0.0189, +0.0800]$	-5.95 $[-6.61, -5.34]$	-13.5 $[-16.4, -10.6]$	-1.5 $[-10.0, +8.2]$
Independent	$+0.59$ $[-0.49, +2.34]$	$+0.0146$ $[-0.0064, +0.0504]$	-1.19 $[-1.58, -0.82]$	-1.7 $[-3.6, +0.1]$	-2.3 $[-9.5, +4.0]$
No-contention	Not evaluated in the reported run				

Note. Entries are CoRefine minus the baseline; brackets are marginal 95% paired-cluster intervals conditional on this run (Section C.1). Positive favors CoRefine for success/utility; negative favors it for costs. The 1,024 settings form 32 task-sharing clusters. No-contention was not run.

Table 6: Paired differences by serving capacity.

Baseline	Success rate (pp)	Utility	Tokens (10^3/task pair)	Mean aggregate flow (s)	P95 task flow (s)
$c_s = 1$					
Fixed	$+1.17$ $[-1.27, +3.81]$	$+0.0584$ $[+0.0097, +0.1116]$	-12.51 $[-13.27, -11.78]$	-22.5 $[-26.6, -19.0]$	-5.5 $[-17.3, +6.3]$
Model-only	$+1.07$ $[-1.86, +4.39]$	$+0.0726$ $[+0.0139, +0.1393]$	-11.25 $[-12.12, -10.45]$	-39.9 $[-43.6, -36.2]$	-26.5 $[-40.3, -10.0]$
Task-1-only	$+1.95$ $[-1.37, +5.57]$	$+0.0553$ $[-0.0118, +0.1281]$	-6.61 $[-7.03, -6.19]$	-9.6 $[-14.0, -6.2]$	-15.1 $[-28.3, -3.8]$
Task-2-only	$+1.86$ $[+0.49, +3.71]$	$+0.0588$ $[+0.0321, +0.0971]$	-5.95 $[-6.61, -5.36]$	-15.8 $[-18.9, -12.6]$	-2.5 $[-12.3, +10.2]$
Independent	$+0.98$ $[0.00, +2.64]$	$+0.0231$ $[+0.0032, +0.0585]$	-1.21 $[-1.60, -0.83]$	-2.4 $[-4.6, -0.5]$	-2.8 $[-12.0, +6.5]$
$c_s = 2$					
Fixed	$+0.68$ $[-1.56, +3.03]$	$+0.0422$ $[-0.0019, +0.0897]$	-12.51 $[-13.29, -11.77]$	-16.0 $[-20.1, -12.5]$	$+0.2$ $[-6.9, +9.7]$
Model-only	-0.10 $[-2.64, +2.83]$	$+0.0322$ $[-0.0202, +0.0922]$	-11.23 $[-12.04, -10.47]$	-22.9 $[-25.9, -20.1]$	-9.8 $[-20.0, +0.6]$
Task-1-only	$+2.05$ $[-1.07, +5.37]$	$+0.0567$ $[-0.0071, +0.1243]$	-6.61 $[-7.04, -6.19]$	-9.1 $[-13.3, -6.2]$	-11.0 $[-24.9, -2.5]$
Task-2-only	$+0.49$ $[-1.07, +2.54]$	$+0.0270$ $[-0.0046, +0.0676]$	-5.94 $[-6.61, -5.34]$	-11.3 $[-14.1, -8.4]$	-1.8 $[-7.5, +7.3]$
Independent	$+0.20$ $[-1.17, +2.15]$	$+0.0061$ $[-0.0217, +0.0457]$	-1.18 $[-1.56, -0.81]$	-1.0 $[-2.8, +0.8]$	-0.7 $[-7.9, +3.7]$

Note. Metrics and conditional intervals follow Table 5. Each capacity pools 512 settings over loads and task-pair categories: 32 four-task clusters and 1,024 task flows.

Table 7: Paired differences by task difficulty.

Baseline	Success rate (pp)	Utility	Tokens (10^3/task pair)	Mean aggregate flow (s)	P95 task flow (s)
H→E					
Fixed	+2.73 [0.00, +6.64]	+0.0749 [+0.0192, +0.1525]	−10.71 [−11.77, −9.70]	−9.5 [−14.7, −4.4]	+13.0 [+0.9, +18.7]
Model-only	+2.73 [−0.39, +7.03]	+0.0871 [+0.0220, +0.1766]	−9.32 [−10.34, −8.37]	−23.1 [−28.6, −17.6]	−4.3 [−18.9, +4.9]
Task-1-only	+4.88 [+0.39, +10.35]	+0.1129 [+0.0216, +0.2255]	−6.08 [−6.58, −5.60]	−9.2 [−14.1, −5.4]	−7.9 [−18.4, −2.5]
Task-2-only	+1.95 [−0.20, +5.66]	+0.0508 [+0.0057, +0.1247]	−4.75 [−5.66, −3.91]	−7.0 [−12.6, −1.4]	+8.0 [−4.1, +15.5]
Independent	+1.17 [−1.17, +4.88]	+0.0289 [−0.0176, +0.1038]	−2.39 [−3.17, −1.64]	−3.1 [−6.7, +0.4]	−4.1 [−12.8, +5.5]
H→H					
Fixed	+1.76 [−4.30, +8.20]	+0.0636 [−0.0561, +0.1922]	−12.03 [−13.19, −10.92]	−16.4 [−26.8, −8.4]	+2.7 [−63.0, +14.8]
Model-only	−0.78 [−8.20, +6.64]	+0.0223 [−0.1256, +0.1730]	−10.76 [−12.09, −9.55]	−27.2 [−32.7, −21.8]	−10.4 [−25.6, +2.2]
Task-1-only	+2.93 [−4.30, +10.35]	+0.0789 [−0.0657, +0.2298]	−7.30 [−8.03, −6.62]	−13.1 [−21.4, −7.1]	−14.6 [−58.8, −3.6]
Task-2-only	+1.95 [−0.59, +5.86]	+0.0498 [−0.0006, +0.1276]	−4.73 [−5.64, −3.89]	−6.0 [−11.2, −0.7]	+10.5 [0.0, +17.7]
Independent	+1.17 [−0.78, +4.69]	+0.0296 [−0.0117, +0.1005]	−2.39 [−3.18, −1.64]	−3.7 [−8.0, +0.1]	−4.3 [−14.8, +5.5]
E→H					
Fixed	−0.78 [−4.69, +3.12]	+0.0238 [−0.0518, +0.1023]	−14.17 [−15.12, −13.24]	−25.3 [−28.8, −22.0]	−17.3 [−21.4, −9.6]
Model-only	+1.17 [−3.32, +6.25]	+0.0765 [−0.0147, +0.1763]	−13.13 [−14.50, −12.00]	−39.9 [−43.8, −36.2]	−38.6 [−43.0, −33.3]
Task-1-only	−0.20 [−4.49, +4.69]	+0.0121 [−0.0723, +0.1086]	−7.07 [−7.78, −6.41]	−8.9 [−16.9, −3.8]	−6.7 [−21.5, +4.0]
Task-2-only	+0.59 [−0.20, +1.76]	+0.0394 [+0.0209, +0.0620]	−7.16 [−7.82, −6.54]	−20.5 [−22.8, −18.3]	−15.5 [−19.7, −8.1]
Independent	0.00 [0.00, 0.00]	0.0000 [0.0000, 0.0000]	0.00 [0.00, 0.00]	0.0 [0.0, 0.0]	0.0 [0.0, 0.0]
E→E					
Fixed	0.00 [−3.12, +2.73]	+0.0389 [−0.0247, +0.0932]	−13.14 [−13.79, −12.51]	−25.7 [−27.8, −23.8]	−20.4 [−23.9, −16.0]
Model-only	−1.17 [−4.69, +1.17]	+0.0237 [−0.0478, +0.0696]	−11.75 [−12.39, −11.13]	−35.4 [−38.5, −32.4]	−38.1 [−42.9, −32.2]
Task-1-only	+0.39 [−4.10, +4.69]	+0.0200 [−0.0701, +0.1050]	−5.98 [−6.40, −5.58]	−6.2 [−7.5, −5.1]	−3.1 [−5.2, +0.1]
Task-2-only	+0.20 [0.00, +0.59]	+0.0316 [+0.0257, +0.0408]	−7.15 [−7.82, −6.53]	−20.6 [−22.9, −18.4]	−19.4 [−23.2, −15.8]
Independent	0.00 [0.00, 0.00]	0.0000 [0.0000, 0.0000]	0.00 [0.00, 0.00]	0.0 [0.0, 0.0]	0.0 [0.0, 0.0]

Note. Metrics and conditional intervals follow Table 5. Each category has 256 settings, 32 task-pair clusters, and 512 task flows. Displayed zeros can reflect rounding or shared outcomes.

Table 8: Paired differences by background load.

Baseline	Success rate (pp)	Utility	Tokens (10^3/task pair)	Mean aggregate flow (s)	P95 task flow (s)
$\rho = 0.20$					
Fixed	$+0.59$ $[-1.76, +3.32]$	$+0.0426$ $[-0.0059, +0.0972]$	-12.52 $[-13.30, -11.77]$	-18.4 $[-22.2, -14.9]$	-2.6 $[-7.9, +4.9]$
Model-only	-0.20 $[-2.93, +2.93]$	$+0.0303$ $[-0.0274, +0.0942]$	-11.19 $[-11.95, -10.46]$	-23.0 $[-26.9, -19.3]$	-13.3 $[-20.8, -4.3]$
Task-1-only	$+2.34$ $[-0.98, +5.86]$	$+0.0627$ $[-0.0033, +0.1336]$	-6.59 $[-7.04, -6.17]$	-9.3 $[-13.5, -6.3]$	-10.0 $[-20.0, -3.1]$
Task-2-only	$+0.78$ $[-0.78, +2.73]$	$+0.0338$ $[-0.0001, +0.0757]$	-5.92 $[-6.59, -5.32]$	-12.2 $[-15.7, -9.0]$	-0.6 $[-7.4, +6.4]$
Independent	$+0.59$ $[-0.78, +2.54]$	$+0.0144$ $[-0.0133, +0.0541]$	-1.19 $[-1.58, -0.81]$	-1.5 $[-3.4, +0.2]$	-3.6 $[-12.0, +3.4]$
$\rho = 0.50$					
Fixed	-0.39 $[-2.93, +2.34]$	$+0.0240$ $[-0.0285, +0.0779]$	-12.51 $[-13.29, -11.78]$	-19.3 $[-24.4, -15.4]$	-1.9 $[-8.5, +13.0]$
Model-only	-0.59 $[-3.52, +2.54]$	$+0.0270$ $[-0.0307, +0.0919]$	-11.27 $[-12.15, -10.47]$	-27.4 $[-30.4, -24.5]$	-11.2 $[-22.4, +2.2]$
Task-1-only	$+1.37$ $[-2.34, +5.27]$	$+0.0432$ $[-0.0323, +0.1211]$	-6.61 $[-7.04, -6.19]$	-9.3 $[-13.5, -6.2]$	-11.9 $[-24.5, -2.8]$
Task-2-only	$+0.78$ $[-0.59, +2.73]$	$+0.0342$ $[+0.0058, +0.0738]$	-5.95 $[-6.61, -5.35]$	-12.6 $[-15.5, -9.8]$	$+0.5$ $[-5.5, +13.8]$
Independent	$+0.78$ $[0.00, +2.34]$	$+0.0191$ $[+0.0013, +0.0528]$	-1.20 $[-1.59, -0.82]$	-2.2 $[-4.8, -0.3]$	-5.0 $[-12.4, +0.9]$
$\rho = 0.90$					
Fixed	$+2.34$ $[+0.20, +4.88]$	$+0.0799$ $[+0.0376, +0.1291]$	-12.50 $[-13.27, -11.77]$	-20.5 $[-24.7, -16.5]$	$+2.3$ $[-14.5, +14.2]$
Model-only	$+2.15$ $[-0.59, +5.27]$	$+0.0924$ $[+0.0351, +0.1549]$	-11.26 $[-12.13, -10.46]$	-38.1 $[-42.0, -34.2]$	-18.5 $[-38.4, -2.4]$
Task-1-only	$+2.73$ $[-0.20, +5.86]$	$+0.0707$ $[+0.0133, +0.1342]$	-6.63 $[-7.05, -6.22]$	-9.4 $[-13.2, -6.6]$	-13.9 $[-28.7, -5.1]$
Task-2-only	$+2.34$ $[+0.59, +4.49]$	$+0.0671$ $[+0.0315, +0.1116]$	-5.96 $[-6.62, -5.36]$	-14.3 $[-18.0, -10.5]$	$+2.8$ $[-14.4, +14.5]$
Independent	$+0.59$ $[-0.78, +2.54]$	$+0.0148$ $[-0.0150, +0.0549]$	-1.19 $[-1.58, -0.82]$	-1.9 $[-4.0, +0.1]$	-4.2 $[-16.5, +11.2]$
$\rho = 0.95$					
Fixed	$+1.17$ $[-1.37, +3.91]$	$+0.0548$ $[+0.0050, +0.1079]$	-12.51 $[-13.28, -11.78]$	-18.9 $[-22.8, -15.3]$	$+0.8$ $[-12.2, +9.8]$
Model-only	$+0.59$ $[-2.73, +4.30]$	$+0.0600$ $[-0.0053, +0.1362]$	-11.24 $[-12.11, -10.44]$	-37.1 $[-40.6, -33.6]$	-19.4 $[-33.9, -7.9]$
Task-1-only	$+1.56$ $[-1.95, +5.27]$	$+0.0473$ $[-0.0239, +0.1240]$	-6.60 $[-7.03, -6.19]$	-9.5 $[-14.8, -5.3]$	-14.5 $[-37.6, -2.0]$
Task-2-only	$+0.78$ $[-0.78, +2.73]$	$+0.0365$ $[+0.0056, +0.0768]$	-5.96 $[-6.62, -5.37]$	-14.9 $[-18.0, -11.8]$	$+1.2$ $[-9.6, +9.5]$
Independent	$+0.39$ $[-0.78, +2.34]$	$+0.0101$ $[-0.0149, +0.0491]$	-1.20 $[-1.59, -0.82]$	-1.1 $[-3.3, +1.0]$	$+1.1$ $[-6.8, +10.0]$

Note. Metrics and conditional intervals follow Table 5. Each load pools 256 settings over capacities and task-pair categories: 32 four-task clusters and 512 task flows.

D Additional Results

D.1 Prefix evidence and independent selection

Independent selects each task's local utility maximizer without joint reranking (Table 9). In each Hard-T1 category, the 20 task pairs with failed arrival screens account for every decision difference: CoRefine uses two rather than four additional samples per writer, retaining the same two-strong-repair suffix. Choices agree at passing checkpoints, including all Easy-T1 cases. CoRefine and Task-1-only also select the same Task 1 continuation throughout.

For Easy-T1 cases, Task 2 has one strong writer, three samples, and one repair round. Each category uses a medium repair model in 208 episodes and a strong model in 48. Thus stable sampling and repair counts do not imply identical model assignments. These traces locate the selection differences without isolating the contention term's contribution.

Table 9: Independent-selection results. Ratios use Fixed in the same task-pair category.

Task-pair category	Success rate (%)	R_A	Token ratio	Flow ratio
H → E	76.37	1.4693	0.528	0.883
H → H	70.31	1.3440	0.510	0.805
E → H	84.38	1.6576	0.240	0.501
E → E	90.43	1.7834	0.220	0.455

D.2 Workflow-role trade-offs

Table 10 separates E→H utility changes into the two tasks' success contributions and saved cost. CoRefine stops Task 1 widening and gives Task 2 one strong writer, three samples, and one repair round in all eight cells; Fixed uses two writers with five samples each. The changes combine sampling, writer composition, and possible repair-model reassignment, rather than isolating $n = 3$ versus $n = 5$.

Table 10: CoRefine–Fixed differences for E→H. Cost contributes $-\alpha\Delta(F_1 + F_2) - \beta\Delta T$; each row has 32 episodes across two hardware platforms. Totals may differ slightly due to rounding.

Cap.	Load	ΔQ_1	ΔQ_2	Cost contribution	ΔR_A
1	Low	0.000	0.031	0.039	0.071
1	Moderate	0.031	-0.125	0.043	-0.051
1	High	0.000	0.031	0.046	0.078
1	Near-sat.	-0.031	0.000	0.043	0.012
2	Low	-0.031	-0.063	0.034	-0.060
2	Moderate	-0.031	-0.063	0.039	-0.054
2	High	0.000	0.063	0.034	0.096
2	Near-sat.	0.031	0.031	0.037	0.099

D.3 Resource demand and the low-load exception

Predicted strong-tier work averages 75.4, 98.9, and 28.8 seconds for Fixed, Model-only, and CoRefine; mean queued-request counts are 3.73, 5.30, and 2.28. The first measures predicted service work rather than GPU busy time; the second includes foreground and background requests. Overall per-task P95 is 53.2 seconds for Fixed and 50.9 for CoRefine, although the Hard-T1 categories can worsen (Table 1).

CoRefine has the highest observed mean utility in seven of eight capacity–load cells. At two replicas and $\rho = 0.20$, Model-only instead obtains 1.5991 versus 1.5779, with success rates of 82.81% versus 80.47%. CoRefine reduces aggregate flow from 40.3 to 25.8 seconds and tokens from 16,830 to 5,681, but the utility gain from saved cost ($+0.0257$) is smaller than the success loss (-0.0469), giving -0.0212 overall. This case illustrates when retaining computation pays off.

D.4 PREFERENCE SENSITIVITY

For baseline b, let ΔQ_b be the difference in mean summed success indicators and $S_{F,b}, S_{T,b}$ the mean aggregate-flow and token savings. Rescoring fixed outcomes gives

$$\Delta R_b(\alpha, \beta) = \Delta Q_b + \alpha S_{F,b} + \beta S_{T,b}. \tag{15}$$

The nominal weights $(\alpha_0, \beta_0) = (10^{-3}, 10^{-6})$ value one additional task success at 1,000 seconds or 10^6 tokens. Savings are positive against the four controls in every category. H→E also has higher observed success; the other categories have the four trade-offs in Table 11. With $(\alpha, \beta) = \gamma(\alpha_0, \beta_0)$, all four crossovers occur by $\gamma = 0.497$. These thresholds compare already executed configurations; they do not measure how the planner changes its choices under new weights.

Table 11: Utility crossover thresholds for fixed outcomes. Entries are the smallest nonnegative scale matching the baseline; larger values favor COREFINE. Omitted comparisons favor COREFINE for all nonnegative weights.

Task-pair category	Control	Joint scale γ Both scaled	α/α_0 $\beta = \beta_0$	β/β_0 $\alpha = \alpha_0$
H → H	Model-only	0.411	0.179	0.000
E → H	Fixed	0.396	0.058	0.000
E → H	Task-1-only	0.245	0.000	0.000
E → E	Model-only	0.497	0.330	0.000

E POST HOC COST CALIBRATION

This separate diagnostic fits cost predictors after the main decisions were frozen. It uses 3,072 distinct episode–Task 2-plan observations from 64 development tasks. Standardized ridge models predict log tokens and log flow from 20 and seven features, respectively, with regularization 1 and 30. Four-fold task-grouped validation over five fold assignments gives in-sample/validation R^2 of $0.9821/0.8973$ for tokens and $0.3018/0.2947$ for flow. Coverage is limited to five configurations, all with a strong writer and one repair round.

The calibrated scorer retains coarse shortlists and success priors, rescales Task 1 costs from the original Fixed prefix, and predicts Task 2 costs with the ridge models. Its flow target retains contention while scoring adds explicit interference, so these charges may overlap. Probe collection and predictor initialization are outside the arrival-planning timing.

Only configurations differing from all six original selections were executed: 18 failed-screen Hard-T1 cases, nine per category. Each removes Task 1's two additional samples per writer and two strong repairs, leaving Task 2 unchanged. Table 12 shows lower resource use but also lower success and utility. All Task 2 identities appeared in training, and both Task 1 rescaling and Task 2 prediction change together. This is a selected development diagnostic, not a held-out policy evaluation.

Table 12: Means for 18 selected development replays; $Q_1 + Q_2 \in [0, 2]$. Aggregate flow sums both tasks.

Variant	$Q_1 + Q_2$	Utility	Tokens	Aggregate flow (s)
Original coarse COREFINE	1.3333	1.2303	9,590	93.43
Calibrated COREFINE	0.9444	0.8904	4,471	49.60