LAMaS: A Modular Platform for Learning Low-Latency Multi-Agent Orchestration

Xi Shi **Mengxin Zheng** **Qian Lou**
LoopLeap Lab, University of Central Florida
{xi320101, mengxin.zheng, qian.lou}@ucf.edu
GitHub: https://github.com/loopleap-lab/LAMaS

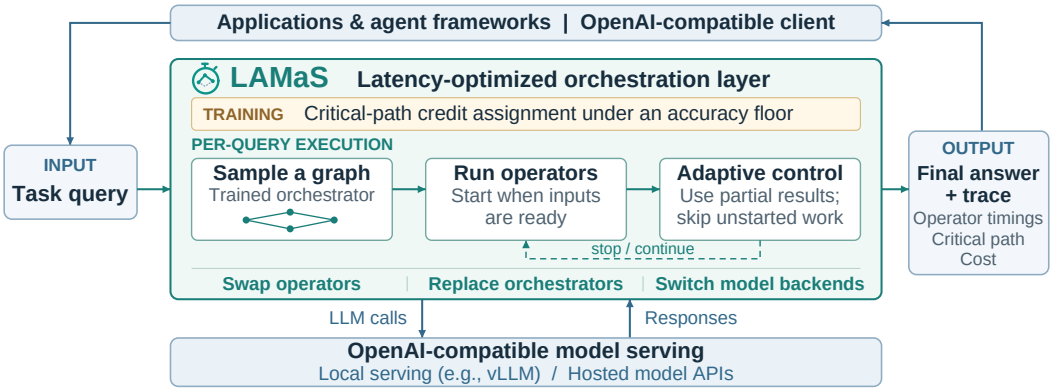

Figure 1: LAMaS connects agent frameworks to model servers and returns an answer and execution trace for each query. We integrate the full pipeline into GDesigner and AgentDropout, retaining each framework's graph builder and retraining the execution controller on its traces.

Abstract

In a multi-agent system (MAS), removing an expensive agent call can save tokens without making the answer arrive sooner: latency depends on the longest chain of dependent calls. We present **L**atency-**A**ware **M**ulti-**a**gent **S**ystem (**LAMaS**), a modular platform for learning low-latency orchestration. A framework can send a query to LAMaS and receive an answer and execution trace, or keep its own graph builder and use the platform's latency-optimized orchestration layer. During training, the layer gives operators on the critical path the greatest latency penalty while targeting a floor on expected accuracy. During inference, a lightweight controller uses partial results to decide whether to skip work that has yet to start. Across four benchmarks, LAMaS reduces end-to-end latency by over 50% relative to the evaluated learning-based MAS baselines while maintaining competitive or better accuracy. GDesigner and AgentDropout retain their graph builders and train the controller on their own traces; both achieve lower latency across the same benchmarks.

1 Introduction

Multi-agent systems (MAS) use specialized agents to extend the reasoning capabilities of large language models (LLMs) [Li et al., 2023, Wu et al., 2024, Hong et al., 2024]. As the number of agents and interactions grows, inference latency becomes a bottleneck [Chen et al., 2025, Shi et al.,

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

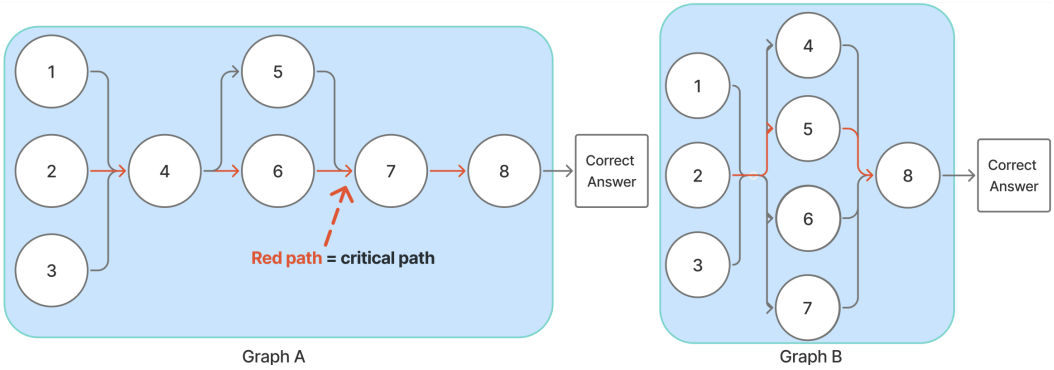

Figure 2: Two execution graphs that produce the same answer at similar cost but differ in latency. Nodes denote agent invocations and arrows denote data flow; red edges mark the critical execution path, the longest chain of dependencies that must complete before an answer is available. Graph B has a shorter critical path and therefore lower wall-clock latency than Graph A.

2026, Kabir et al., 2026a,b]: tasks in computer use [Abhyankar et al., 2026] and autonomous research [Kang et al., 2025] can take 10–60 minutes. Quantization can speed up individual model calls [Lou et al., 2020, Zhao et al., 2021], but the time a user waits also depends on which calls must finish before others can start.

We introduce **L**atency-**A**ware **M**ulti-**a**gent **S**ystem (**LAMaS**), a modular platform that separates an agent framework's choice of operators from how it learns and executes low-latency workflows (Figure 1). Applications can submit queries through an OpenAI-compatible endpoint. Framework developers can instead keep their graph builder and adopt the orchestration layer's training objective and execution controller (Section 5).

Existing methods learn to orchestrate agents for accuracy and inference cost [Zhang et al., 2025a, Wang et al., 2025c, Zhang et al., 2025c,d, Wang et al., 2025b,a, Yao et al., 2025]. Cost adds up across calls; latency follows the *critical path*, the longest chain of dependent calls measured in time. Figure 2 shows why the distinction matters: two graphs produce the same answer at similar cost, but the one with the shorter critical path finishes sooner.

Training and execution call for different decisions. When learning a graph, a uniform latency penalty gives the same signal to operators that delay the answer and those that finish in parallel. LAMaS assigns credit according to each operator's contribution to the critical path and uses Lagrangian relaxation to optimize latency and cost subject to a floor on expected accuracy. At inference time, the graph alone cannot anticipate agreement among early answers. As operators finish, a lightweight controller scores the partial execution under the same objective and decides whether to start more work.

We evaluate LAMaS on GSM8K [Cobbe et al., 2021], HumanEval [Chen et al., 2021], MATH [Hendrycks et al., 2021], and MMLU-Pro [Wang et al., 2024] against MaAS [Zhang et al., 2025a], GDesigner [Zhang et al., 2025c], and AgentDropout [Wang et al., 2025c]. LAMaS has the lowest latency of the evaluated MAS on all four benchmarks, with competitive or better accuracy. We also integrate the layer into GDesigner and AgentDropout: each retains its own graph builder, trains with critical-path credit, and fits the controller to its own execution traces. Both integrations reduce latency across all four benchmarks (Table 3).

2 Related Work

LLM-Based Multi-Agent Systems. Multi-Agent Systems built on LLMs decompose complex problems into subtasks handled by specialized agents. Early frameworks such as CAMEL [Li et al., 2023], AutoGen [Wu et al., 2024], and MetaGPT [Hong et al., 2024] demonstrated the potential of collaborative problem-solving through role-playing and structured communication. However, these approaches rely on static, hand-crafted topologies that are labor-intensive to design and difficult to adapt across tasks and difficulty levels.

Automated Multi-Agent Orchestration. To overcome the rigidity of manual designs, recent work treats orchestration as a search or optimization problem. AnyMAC [Wang et al., 2025a], SeqCV [Yao et al., 2025], and Gated-Memory Routing [Rajib et al., 2026a] dynamically adjust chain length per query. Aflow [Zhang et al., 2025d] and EvoAgentX [Wang et al., 2025b] search over execution topologies using evolutionary or flow-based methods. MaAS [Zhang et al., 2025a] models the search space as a probabilistic agentic supernet and trains an orchestrator via policy gradient. AgentPrune [Zhang et al., 2025b] prunes redundant communication messages to obtain more economical communication topologies, while AgentDropout [Wang et al., 2025c] dynamically eliminates redundant agents and communication to improve token efficiency. GDesigner [Zhang et al., 2025c] uses graph neural networks to architect communication topologies. These methods primarily optimize task performance, token efficiency, or inference cost. In contrast, latency is a graph-level systems objective whose bottlenecks are concentrated on the critical execution path, making it difficult to optimize with uniform additive rewards alone. INFRAMIND [Kabir et al., 2026b] conditions planning, per-step routing, and scheduling on the runtime state of the serving infrastructure. Orthogonally, systems-level latency management methods such as Kairos [Chen et al., 2025], HW-Router [Kabir et al., 2026a], and AgentServeSim [Rajib et al., 2026b] optimize serving, batching, or runtime scheduling, and KV-cache reuse across contexts and model checkpoints [Shi et al., 2026] avoids repeated computation on shared text; both are complementary to our focus on learning latency-aware orchestration policies.

Latent Communication and Agentic Routing. LatentMAS [Zou et al., 2026] is a training-free method that replaces text-based communication among agents with the exchange of latent (hidden-state) representations, reducing decoding and communication overhead. GraphPlanner [Feng et al., 2026] performs agentic routing, selecting an (LLM backbone, agent role) pair at each step to improve accuracy and reduce computational cost, which accumulates additively over the steps of the resulting trajectory. LAMaS differs from both in problem setting, graph construction, execution control, and latency optimization: it samples operator execution graphs and optimizes their critical-path latency under an explicit accuracy floor, assigns latency credit to operators according to path criticality, and adds an inference-time controller that eliminates operators not yet launched. Because an operator off the critical path adds cost but not latency, additive cost and wall-clock latency are distinct objectives; latent communication, in turn, reduces per-operator overhead rather than selecting which operators to execute, and is therefore complementary to our approach. LLM routers such as R2-Router [Xue et al., 2026], which jointly select an LLM and an output-length budget for each query, are likewise complementary.

Model-Level Efficiency. Model compression reduces the cost of each model invocation, for example through mixed-precision quantization [Lou et al., 2020, Zhao et al., 2021], weighted low-rank factorization [Hsu et al., 2022], and dictionary lookups that replace dense weight matrices [Lou et al., 2022a,b]. These model-level techniques are complementary to workflow-level latency optimization such as LAMaS, which targets the critical execution path across agent calls.

3 Methodology

3.1 Problem Setting

We consider a query-dependent multi-agent system (MAS) that dynamically composes a set of *agentic operators* to solve an input problem. An agentic operator is the basic execution unit, representing a composite agent invocation that may involve multiple LLM calls and external tool usage. Given a query x, the orchestrator constructs an execution graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a directed acyclic graph (DAG) where each node $o \in \mathcal{V}$ is an operator invocation. The edge set $\mathcal{E}$ is induced by operator I/O flow: an edge $(o_i, o_j) \in \mathcal{E}$ exists when the output produced by o_i is used as input by o_j during execution.

We define the *critical path* as the longest path in $\mathcal{G}$ measured by execution time:

$$T = \max_{p \in \text{Paths}(\mathcal{G})} \sum_{o \in p} t(o), \tag{1}$$

where $t(o)$ denotes the execution time of operator o, and $\text{Paths}(\mathcal{G})$ denotes all source-to-sink paths in $\mathcal{G}$. The critical path determines the end-to-end latency: operators off the critical path do not affect

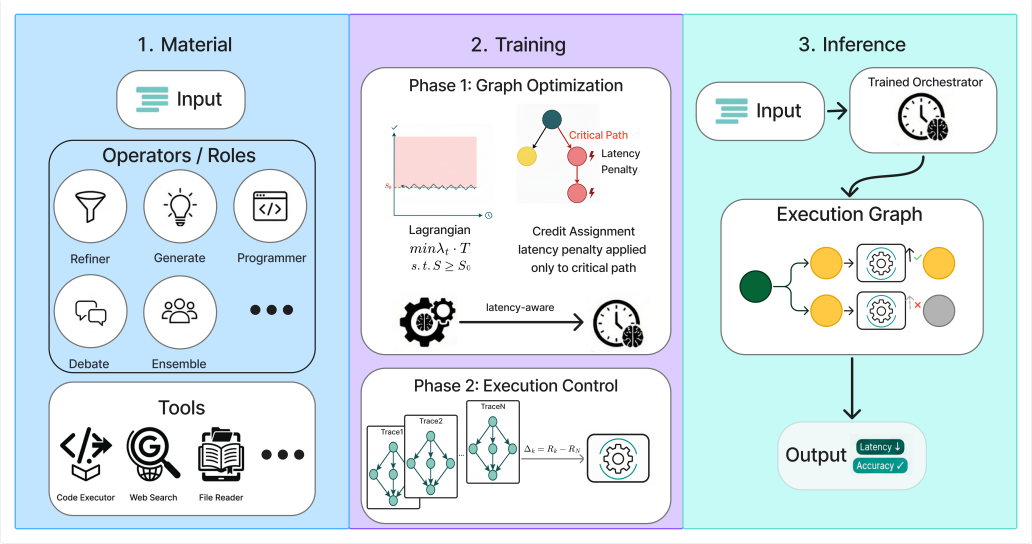

Figure 3: Training and inference in the LAMaS orchestration layer. **(1) Material:** A library of reusable operators and tools. **(2) Training:** Phase 1 performs latency-aware graph optimization via Lagrangian-constrained learning and critical-path credit assignment, where S_0 denotes the target accuracy floor enforced during optimization. Phase 2 trains a lightweight execution controller on the orchestrator's traces to predict the control advantage. **(3) Inference:** The trained orchestrator first constructs the execution graph; the execution controller then performs online execution control, monitoring operator completions and adaptively eliminating redundant future agent interactions.

wall-clock time. In contrast, the total execution cost accumulates additively across all operators: $C = \sum_{o \in \mathcal{V}} c(o)$. Latency and cost therefore diverge.

Our goal is to learn an orchestrator that preserves task accuracy while minimizing end-to-end latency.

Execution and final-answer semantics. Operators in $\mathcal{G}$ produce *candidate solutions* rather than final answers: every completed answer-producing operator contributes its output(s) to a shared candidate pool. The final answer is always emitted by a designated terminal stage (the sink of $\mathcal{G}$) applied to this pool: when several candidates are present, it selects one (e.g., via the self-consistency ensemble); when only one is present, that candidate passes through unchanged. This holds for both full and early-stopped executions. When the execution controller (Section 3.6) stops an execution early, it only prevents operators that have not been launched from starting; operators already launched finish normally, and the terminal stage then runs once on the resulting candidates.

3.2 Orchestrator Architecture

LAMaS represents the orchestrator as a policy network over an *agentic supernet*—a probabilistic DAG over a fixed set of candidate operators $\mathcal{O}$ (a formulation also adopted by Zhang et al. [2025a]). The latency-aware training and credit assignment introduced below are agnostic to the specific architectural choice and apply to any orchestrator that produces a differentiable distribution over execution graphs. The orchestrator constructs $\mathcal{G}$ by sampling operators step-by-step, with the generation probability factorized as:

$$P_\theta(\mathcal{G}|x) = \prod_{\ell=1}^{L} \pi_\theta(\mathcal{V}_\ell \mid x, \mathcal{G}_{1:\ell-1}), \tag{2}$$

where $\mathcal{V}_\ell \subseteq \mathcal{O}$ is the operator subset selected at step ℓ. The network produces a query-aware operator-level policy via softmax over activation logits; we denote the resulting probability of operator o by $s_o := p_\theta(o \mid x, \mathcal{G}_{1:\ell-1})$. Operators are sampled from this policy sequentially without replacement until their cumulative probability first exceeds a threshold τ:

$$\mathcal{V}_\ell = \{o_1, \ldots, o_k\}, \quad k = \min\Big\{j : \textstyle\sum_{i=1}^{j} s_{o_i} > \tau\Big\}, \tag{3}$$

enabling the architecture to dynamically adjust width and depth per query.

3.3 Training-Time Constrained Optimization

Existing multi-agent orchestration methods typically combine accuracy and cost into a single weighted reward (e.g., $R = S - \lambda_c C$). However, as discussed in Section 1, latency and cost are structurally different objectives. Because latency is governed by the critical path, uniform penalties provide weak operator-level attribution; without an explicit accuracy floor, latency-driven optimization can degrade task performance; and a graph fixed at training time cannot leverage execution-time evidence. LAMaS addresses these via critical-path-aware credit assignment, an accuracy floor enforced by Lagrangian relaxation, and a lightweight execution-time controller. We therefore formulate MAS orchestration as a constrained optimization:

$$\min_{\theta} \ \mathbb{E}[\lambda_t T + \lambda_c C] \,, \quad \text{s.t.} \quad \mathbb{E}[S] \geq S_0, \tag{4}$$

where $S \in \{0, 1\}$ is the task score, C is the execution cost in USD per query, T is the end-to-end wall-clock latency in seconds per query, and λ_t, λ_c are fixed weights controlling the relative importance of latency and cost. Here, S_0 is a fixed target accuracy floor on $\mathbb{E}[S]$, rather than a per-instance requirement, and in practice is set to the architecture's reference accuracy (see the appendix for sensitivity to different floors).

The accuracy floor S_0 acts as a "do-no-harm" constraint: LAMaS reduces latency without falling below the reference accuracy of the underlying architecture. In our main experiments, S_0 is set to the validation accuracy of the same orchestrator trained with $\lambda_t = 0$; we analyze sensitivity to S_0 in the appendix. We solve the constrained problem via Lagrangian relaxation with an adaptive dual variable:

$$R = \lambda_S \frac{S}{S_0} - \lambda_t T - \lambda_c C, \tag{5}$$

where λ_S is the dual variable updated via projected gradient ascent:

$$\lambda_S \leftarrow \max\left(0, \ \lambda_S + \eta \left(1 - \frac{\bar{S}}{S_0}\right)\right), \tag{6}$$

where $\bar{S}$ denotes the batch-averaged accuracy and η is the dual learning rate. When accuracy drops below S_0, λ_S automatically increases to prioritize correctness; when accuracy is satisfied, λ_S decreases to allow more aggressive latency reduction. This adaptive coupling removes the need to hand-tune a fixed accuracy–latency trade-off weight.

3.4 Critical-Path-Aware Credit Assignment

End-to-end latency is determined by the critical path. Applying the latency penalty uniformly to all operators creates a mismatch in operator-level credit assignment: operators that have little or no influence on T receive the same gradient as bottleneck operators on the critical path. We therefore use a soft bottleneck-aware weighting that preserves full penalty on the realized critical path while assigning attenuated latency credit to operators on near-critical paths.

For each operator o in the realized execution graph $\mathcal{G}$, let $\ell(o)$ denote the latency of the longest path passing through o, and define the path-criticality weight

$$w(o) \ = \ \frac{\ell(o)}{T} \ \in \ [0, 1]. \tag{7}$$

By construction $w(o) = 1$ exactly when o lies on the critical path, $w(o) \approx 1$ for operators on near-critical paths, and $w(o) \ll 1$ for operators whose longest containing path is far below T. We assign operator-level rewards as

$$R(o) \ = \ \lambda_S \frac{S}{S_0} \ - \ \lambda_t \, w(o) \, T \ - \ \lambda_c C, \tag{8}$$

so each operator receives a fraction $w(o)$ of the global latency penalty $-\lambda_t T$: critical-path operators are penalized in full, near-critical operators receive a smoothly attenuated share, and operators with negligible $w(o)$ get effectively no latency signal. This biases the gradient toward bottleneck operators while preserving softer latency attribution for near-critical paths; the values $\ell(o)$, and hence $w(o)$, are obtained alongside the critical path itself via a single forward/backward pass over $\mathcal{G}$.

3.5 Learning Objective

The orchestrator is trained with policy gradient optimization. Let ξ denote the trajectory of operator selections; the loss is

$$\mathcal{L}(\theta) = -\mathbb{E}_{\xi \sim \pi_\theta} \left[\sum_{\ell=1}^{L} \sum_{o \in \mathcal{V}_\ell} R(o) \log p_\theta(o \mid x, \mathcal{G}_{1:\ell-1}) \right]. \tag{9}$$

To reduce variance, we normalize rewards using exponential moving averages (EMA). Specifically, we maintain running estimates $\hat{\mu}$ and $\hat{\sigma}^2$ of the mean and variance of $R(o)$ with momentum 0.95, and normalize each reward as $(R(o) - \hat{\mu})/\hat{\sigma}$ before computing the policy gradient; this is more stable than per-batch normalization for small batch sizes.

3.6 Inference-Time Execution Control

Because the orchestrator can activate multiple operators per step (Section 3.2), multiple candidate outputs become available as an execution proceeds, so answer consensus among completed candidates is well-defined at intermediate decision points. The orchestrator commits to the full execution graph based solely on the query x, but each operator completion reveals new information—output content, emerging answer consensus, and elapsed time—unavailable at construction time [Rajib et al., 2026a]. LAMaS therefore augments graph construction with an execution-time controller, implemented as a lightweight learned module. This controller can be viewed as performing an online elimination of redundant future agent interactions: at each operator completion, it evaluates whether the current partial execution should be retained or whether additional agent interactions remain worthwhile, by predicting the *control advantage*—how much better (or worse) the current partial execution would be compared to running all operators:

$$\Delta_k = R_k - R_N, \tag{10}$$

where R_k is the Lagrangian reward (Eq. 5) evaluated at the k-th operator completion and R_N is the reward under full execution. R_k is a *counterfactual* stop-at-k evaluation: the reward that would be obtained if execution stopped right after the k-th operator completion, i.e., if the final answer were produced from the candidates available at that point. It is computed only offline, from the full-execution traces recorded for controller training (described below); at inference time R_k is never computed, and a final answer is produced exactly once, when execution actually stops. $\Delta_k > 0$ means the current partial execution is preferable because it preserves accuracy while reducing latency; $\Delta_k < 0$ means additional agent interactions remain beneficial.

The controller is a small MLP that takes execution-state features as input and predicts $\hat{\Delta}_k$:

$$\hat{\Delta}_k = \text{Controller}_\phi \left(\frac{n_{\text{completed}}}{n_{\text{total}}},\ \frac{\max_\text{agree}}{n_{\text{cand}}},\ \text{Compress}(q) \right), \tag{11}$$

where $n_{\text{completed}}$ is the number of operators that have finished execution at the current decision point, n_{total} is the total number of operators scheduled in the sampled execution graph, n_{cand} is the number of candidate answers extracted from the completed operators, and $\max_\text{agree}$ is the size of the largest cluster of mutually agreeing answers among these candidates. The first feature thus measures execution progress and the second the current rate of agreement (answer consensus); $\text{Compress}(q)$ is a learned low-dimensional projection of the query embedding for difficulty awareness. We define consensus via task-specific semantic equivalence of the current operator outputs: for GSM8K and MATH, we extract the boxed answer (or, if absent, the final numeric answer) and compare exact matches; for MMLU-Pro, we extract the predicted option letter and compare exact matches; for HumanEval, we regard two outputs as agreeing if they induce the same pass/fail outcomes on the test cases available during execution (i.e., the same execution feedback used by the **Test** operator). The predicted advantage is converted to a selection score via:

$$p(\text{select current}) = \sigma(\hat{\Delta}_k / \sigma_\Delta), \tag{12}$$

where $\sigma_\Delta = \text{std}(\{\Delta_k \mid \Delta_k > 0\})$ is computed from training data. At inference time, we sample the retain/eliminate decision from this Bernoulli score at each decision point.

Controller training. The execution controller is trained *after* the orchestrator converges, using execution traces collected by running the frozen orchestrator on the training set. For each query, all operators are executed to completion, and Δ_k is computed for every decision point k. The controller is trained via MSE regression: $\min_\phi \mathbb{E}[(\hat{\Delta}_k - \Delta_k)^2]$. Since R_k uses the same Lagrangian reward as the orchestrator, the controller inherits the orchestrator's latency-aware objective under the same accuracy floor—in particular, the dual variable λ_S encodes how difficult accuracy is to maintain, automatically calibrating the controller's aggressiveness.

The full training and inference procedure is summarized in Appendix C.3 (Algorithm 1). Phase 1 trains the orchestrator with Lagrangian-constrained policy gradient and critical-path credit assignment. Phase 2 trains the execution controller on traces from the frozen orchestrator to predict the control advantage.

4 Experiments

4.1 Experimental Setup

Benchmarks & Tasks. We evaluate our method on four benchmarks spanning three task categories. For HUMANEVAL, GSM8K, and MATH, we follow the same training and evaluation splits as MaAS [Zhang et al., 2025a]; for MMLU-PRO [Wang et al., 2024], we construct our own stratified train/test split from the official test set. Detailed split statistics are provided in Appendix B.3. For **code generation**, we use HUMANEVAL, which measures functional correctness of Python programs generated from natural language descriptions. For **mathematical reasoning**, we use GSM8K and MATH, which consist of grade-school and competition-level math problems requiring multi-step reasoning. For **knowledge reasoning**, we use MMLU-PRO, which covers diverse professional-level questions requiring multi-step reasoning.

Baselines. We compare against two categories of baselines: (i) **non-MAS prompting strategies**—**Generate** (direct single-step answer), **CoT** [Wei et al., 2022], and **SC** [Wang et al., 2023]; and (ii) **learned MAS orchestrators**—**GDesigner** [Zhang et al., 2025c], **AgentDropout** [Wang et al., 2025c], and **MaAS** [Zhang et al., 2025a]. Detailed per-baseline configurations are provided in Appendix B.4.

Evaluation Metrics. We evaluate all methods using three metrics: task performance, API cost, and latency. For task performance, we report pass@1 on HUMANEVAL, and accuracy on GSM8K, MATH, and MMLU-PRO. API cost is measured as the average per-query monetary cost (in units of $\times 10^{-4}$ USD) incurred by LLM API calls. For latency, we directly measure the end-to-end wall-clock execution time, reported as the average latency per query (in seconds). All methods in Tables 1–3 that use API-served backbones are evaluated under the same API provider and execution runtime, without response caching, and latency is averaged over 3 independent runs. The end-to-end latency is determined by the critical execution path as defined in Section 3.1. We also report CP Tokens, the average number of output tokens generated along the critical execution path, as a complementary structural indicator of critical-path workload.

Implementation Details. We use the closed-source LLM `gpt-4o-mini-0718` [OpenAI, 2024]. The latency penalty is set to $\lambda_t = 0.005$, the activation threshold to $\tau = 0.3$, and the supernet has $L = 4$ layers; we report sensitivity to each of these in the appendix. The target accuracy floor S_0 is set to the validation accuracy of the same orchestrator trained without latency optimization (i.e., $\lambda_t = 0$), so the constraint measures whether latency reduction is achieved while maintaining the reference accuracy level of the underlying architecture. Remaining training hyperparameters are listed in Appendix B.2. All results are averaged over 3 independent runs.

4.2 Result Analysis

We analyze the results from three perspectives: evaluated learning-based MAS baselines, the full set of baselines, and component-wise ablations.

Comparison with evaluated learning-based MAS baselines. Table 1 shows that LAMaS consistently reduces end-to-end latency by 55.6–75.6% relative to the strongest evaluated learning-based MAS baselines while maintaining competitive or better accuracy. The reduction co-occurs with

Table 1: Comparison with non-MAS and MAS baselines across four benchmarks. Each cell reports mean $\pm$ standard deviation across 3 independent runs. For MAS methods, the best mean in each column per dataset is shown in **bold**. Cost is reported in units of $\times 10^{-4}$ USD. Latency is the average end-to-end wall-clock time per query (in seconds). CP Tokens denotes the average number of output tokens on the critical execution path.

Dataset	Method	Acc (%) ↑	Cost ($\times 10^{-4}$) ↓	Latency (s) ↓	CP Tokens ↓
GSM8K	Generate	$87.46_{\pm 0.20}$	$2.87_{\pm 0.05}$	$5.43_{\pm 0.18}$	$280_{\pm 8}$
	CoT	$88.97_{\pm 0.43}$	$4.53_{\pm 0.04}$	$6.92_{\pm 0.21}$	$331_{\pm 15}$
	SC	$89.86_{\pm 0.19}$	$18.61_{\pm 0.09}$	$10.64_{\pm 0.27}$	$591_{\pm 22}$
	GDesigner	$93.30_{\pm 0.93}$	$17.86_{\pm 0.28}$	$32.31_{\pm 1.07}$	$1624_{\pm 29}$
	AgentDropout	$93.11_{\pm 0.27}$	$53.81_{\pm 6.78}$	$81.24_{\pm 12.74}$	$4098_{\pm 538}$
	MaAS	$93.36_{\pm 0.72}$	$31.52_{\pm 0.78}$	$48.11_{\pm 3.49}$	$2501_{\pm 53}$
	LAMaS	$\mathbf{93.65_{\pm 0.41}}$	$\mathbf{7.71_{\pm 0.94}}$	$\mathbf{11.73_{\pm 1.13}}$	$\mathbf{622_{\pm 87}}$
HumanEval	Generate	$84.73_{\pm 1.32}$	$5.11_{\pm 0.06}$	$10.23_{\pm 0.71}$	$513_{\pm 31}$
	CoT	$85.50_{\pm 3.05}$	$7.24_{\pm 0.32}$	$14.87_{\pm 1.04}$	$780_{\pm 47}$
	SC	$86.26_{\pm 0.76}$	$28.15_{\pm 0.13}$	$20.41_{\pm 0.57}$	$1105_{\pm 38}$
	GDesigner	$85.50_{\pm 2.02}$	$19.86_{\pm 2.46}$	$49.68_{\pm 5.62}$	$2983_{\pm 489}$
	AgentDropout	$86.01_{\pm 2.20}$	$27.73_{\pm 1.32}$	$51.49_{\pm 6.93}$	$3043_{\pm 218}$
	MaAS	$93.38_{\pm 1.17}$	$28.18_{\pm 1.21}$	$47.32_{\pm 2.93}$	$2708_{\pm 109}$
	LAMaS	$\mathbf{95.42_{\pm 1.53}}$	$\mathbf{12.92_{\pm 1.16}}$	$\mathbf{21.00_{\pm 2.84}}$	$\mathbf{1180_{\pm 187}}$
MATH	Generate	$50.62_{\pm 1.09}$	$6.38_{\pm 0.14}$	$10.17_{\pm 0.34}$	$491_{\pm 19}$
	CoT	$51.23_{\pm 1.15}$	$10.24_{\pm 0.07}$	$21.03_{\pm 0.31}$	$992_{\pm 21}$
	SC	$51.85_{\pm 1.23}$	$42.41_{\pm 0.27}$	$27.62_{\pm 0.41}$	$1340_{\pm 24}$
	GDesigner	$50.82_{\pm 0.71}$	$30.17_{\pm 2.27}$	$60.71_{\pm 5.83}$	$3155_{\pm 246}$
	AgentDropout	$52.47_{\pm 0.94}$	$97.23_{\pm 8.61}$	$176.42_{\pm 19.18}$	$8693_{\pm 974}$
	MaAS	$52.67_{\pm 0.82}$	$65.39_{\pm 1.19}$	$109.87_{\pm 5.92}$	$5602_{\pm 116}$
	LAMaS	$\mathbf{53.29_{\pm 1.25}}$	$\mathbf{28.74_{\pm 3.05}}$	$\mathbf{28.41_{\pm 4.27}}$	$\mathbf{1452_{\pm 232}}$
MMLU-Pro	Generate	$54.47_{\pm 1.14}$	$1.05_{\pm 0.08}$	$2.68_{\pm 0.22}$	$136_{\pm 7}$
	CoT	$56.80_{\pm 3.03}$	$2.41_{\pm 0.15}$	$6.27_{\pm 0.46}$	$361_{\pm 14}$
	SC	$58.67_{\pm 1.21}$	$14.88_{\pm 0.18}$	$10.53_{\pm 0.49}$	$620_{\pm 12}$
	GDesigner	$64.27_{\pm 3.63}$	$25.03_{\pm 0.42}$	$46.66_{\pm 2.47}$	$2750_{\pm 41}$
	AgentDropout	$61.87_{\pm 2.31}$	$76.13_{\pm 1.41}$	$119.33_{\pm 14.68}$	$6560_{\pm 561}$
	MaAS	$65.40_{\pm 3.08}$	$30.37_{\pm 2.56}$	$49.12_{\pm 5.13}$	$2985_{\pm 151}$
	LAMaS	$\mathbf{66.27_{\pm 1.96}}$	$\mathbf{10.93_{\pm 1.39}}$	$\mathbf{14.81_{\pm 1.92}}$	$\mathbf{845_{\pm 96}}$

substantially lower CP Tokens (e.g., 622 vs. 2501 on GSM8K), indicating that the orchestrator produces graphs with shorter critical paths under the latency objective; the inference-time controller further trims execution once a confident answer emerges.

Comparison with baselines. Non-MAS methods (**Generate**, **CoT**, **SC**) remain faster because of their simplicity but consistently underperform in task accuracy. Among MAS methods, **GDesigner** and **AgentDropout** incur substantially higher latency without corresponding accuracy gains, whereas LAMaS achieves the lowest latency across all four benchmarks while maintaining comparable accuracy. Across MAS methods and benchmarks, CP Tokens are strongly correlated with measured wall-clock latency (Appendix C.1), providing complementary evidence that LAMaS reduces critical-path workload together with end-to-end latency. Appendix F.1 further compares LAMaS with single-agent agentic baselines that add tool use or self-correction (tool-augmented CoT, test-time repair, and Self-Refine); LAMaS remains 3.86–9.14 pp more accurate than these baselines.

4.3 Ablation Study

We study four variants of LAMaS: removing CP credit assignment, removing the inference-time controller, removing the latency objective, and a naive "add latency to the reward" baseline. *w/o CP Credit* forces the latency penalty to apply uniformly to all operators rather than weighting it by path criticality; *w/o controller* removes the execution controller entirely and relies solely on training-time optimization; *w/o latency obj.* sets $\lambda_t = 0$, training the orchestrator without any latency objective; *naive lat. only* keeps only a bare uniform latency term in the training objective, with no S_0 accuracy

Table 2: Ablation study of inference-time execution control, critical-path (CP) credit assignment, and the latency objective across four benchmarks. Cost is in units of $\times 10^{-4}$ USD; latency is end-to-end wall-clock time per query (in seconds).

Dataset	Method	Acc (%) ↑	Cost ($\times 10^{-4}$) ↓	Latency (s) ↓
GSM8K	LAMaS	**93.65**	**7.71**	**11.73**
	w/o controller	93.27	9.43	13.84
	w/o CP Credit	92.73	8.71	12.93
	w/o latency obj. (λ_t=0)	93.59	9.71	15.83
	naive lat. only	91.85	9.86	15.52
HumanEval	LAMaS	95.42	**12.92**	**21.00**
	w/o controller	95.17	16.18	24.47
	w/o CP Credit	94.40	14.83	23.43
	w/o latency obj. (λ_t=0)	**95.67**	17.42	35.62
	naive lat. only	93.13	18.07	31.84
MATH	LAMaS	**53.29**	**28.74**	**28.41**
	w/o controller	53.02	34.51	33.46
	w/o CP Credit	52.26	32.18	31.27
	w/o latency obj. (λ_t=0)	53.16	36.84	35.27
	naive lat. only	51.99	37.21	34.27
MMLU-Pro	LAMaS	66.27	**10.93**	**14.81**
	w/o controller	66.00	12.68	17.94
	w/o CP Credit	65.40	11.83	16.18
	w/o latency obj. (λ_t=0)	**66.33**	13.05	22.46
	naive lat. only	60.93	13.74	20.91

floor, no CP-aware credit assignment, and no inference controller—the simplest "add latency to the reward" baseline.

Table 2 shows consistent trends across all four benchmarks. Removing the controller preserves accuracy within 0.38 pp but increases latency by 16.5–21.1% and cost by 16.0–25.2%, confirming the importance of inference-time execution control. Removing CP credit assignment also degrades all three metrics, supporting path-criticality-weighted latency attribution rather than a uniform penalty over all operators. Removing the latency objective entirely (λ_t=0) leaves accuracy essentially unchanged (within ± 0.25 pp of LAMaS) but inflates latency by 24–70% and cost by 19–35%, confirming that the constrained latency objective reshapes the execution graph for efficiency while maintaining the reference accuracy level of the underlying architecture. Finally, the *naive lat. only* variant drops accuracy by 1.30–5.34 pp relative to LAMaS while yielding only 2–11% latency savings over λ_t=0 and no cost reduction, showing that simply adding a latency term provides only limited gains. Taken together, these ablations show that LAMaS combines path-selective latency pressure with an explicit accuracy floor and execution-time control rather than simply shrinking computation.

4.4 Transferability and Generality of LAMaS Mechanisms

The full LAMaS pipeline—the training-time graph optimization, the critical-path credit assignment, and the inference-time execution controller—is not tied to a specific orchestrator architecture. To verify its broader applicability, we integrate the entire pipeline into two other learning-based MAS methods, GDesigner [Zhang et al., 2025c] and AgentDropout [Wang et al., 2025c], without redesigning their original orchestration frameworks.

Across all four benchmarks and both architectures, integrating the full LAMaS pipeline consistently reduces latency by 27–39% while preserving or slightly improving accuracy within typical 3-run noise. These results confirm that latency-aware graph optimization together with inference-time execution control transfers effectively to other learning-based MAS architectures. The reductions are smaller than in the LAMaS-native setting (Table 1), consistent with the fact that GDesigner and AgentDropout retain their original orchestration policies: the LAMaS pipeline contributes the latency-aware credit signal and inference-time control rather than replacing their graph construction. Together with Appendix E, this suggests that the LAMaS pipeline remains effective beyond a single orchestration architecture and extends to the additional backbone and serving configurations we evaluate. Appendix C.4 further shows that the orchestrator can still reasonably select an operator not

Table 3: Transferability of the full LAMaS pipeline (training-time graph optimization, critical-path credit assignment, and inference-time execution control) to other learning-based MAS architectures. The controller is retrained on each architecture's own traces (Phase 2 of the LAMaS training procedure). Latency is the average end-to-end wall-clock time per query (in seconds). CP Tokens denotes the average critical-path output tokens.

Dataset	Method	Acc (%) ↑	Cost ($\times 10^{-4}$) ↓	Latency (s) ↓	CP Tokens ↓
GSM8K	GDesigner	93.30	17.86	32.31	1624
	GDesigner+LAMaS	**93.27**	**12.08**	**20.55**	**1138**
	AgentDropout	93.11	53.81	81.24	4098
	AgentDropout+LAMaS	**93.18**	**39.67**	**58.14**	**2989**
HumanEval	GDesigner	85.50	19.86	49.68	2983
	GDesigner+LAMaS	**85.75**	**12.93**	**30.85**	**1571**
	AgentDropout	86.01	27.73	51.49	3043
	AgentDropout+LAMaS	**86.26**	**17.98**	**36.62**	**2200**
MATH	GDesigner	50.82	30.17	60.71	3155
	GDesigner+LAMaS	**51.23**	**20.03**	**39.93**	**2118**
	AgentDropout	52.47	97.23	176.42	8693
	AgentDropout+LAMaS	**52.40**	**70.35**	**128.34**	**6311**
MMLU-Pro	GDesigner	64.27	25.03	46.66	2750
	GDesigner+LAMaS	**64.33**	**17.03**	**28.27**	**1548**
	AgentDropout	61.87	76.13	119.33	6560
	AgentDropout+LAMaS	**62.20**	**56.23**	**82.27**	**4628**

exposed during training. Appendix F.2 additionally compares GDesigner+LAMaS with two naive inference-time latency-cutting rules applied to GDesigner; neither rule matches its accuracy–latency trade-off.

5 The LAMaS Platform

An application can use LAMaS through its OpenAI-compatible endpoint by changing the server address (Figure 1). For each query, the orchestrator selects a graph of operators. The runtime starts each operator when its inputs are ready and returns the final answer with a trace of operator timings, the critical path, and cost. Operators and graph builders are replaceable; model calls can use a local server such as vLLM or a hosted OpenAI-compatible API.

A framework that builds its own graphs can reuse the orchestration layer directly. In GDesigner and AgentDropout, we train the existing graph policy with the accuracy constraint and critical-path credit. Once that policy is frozen, its execution traces supply the examples for training the stop controller. This preserves each framework's graph construction while letting partial results determine whether the remaining operators should run (Table 3).

6 Conclusion

LAMaS makes low-latency orchestration reusable across multi-agent systems. Its training objective targets the critical path under an accuracy floor, and its controller uses partial results to skip future calls. Across four benchmarks, LAMaS reduces latency by over 50% relative to the evaluated learning-based MAS baselines with competitive or better accuracy. GDesigner and AgentDropout also reduce latency while retaining their own graph builders.

Acknowledgments and Disclosure of Funding

This work was supported in part by the National Science Foundation under Award No. 2544047. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Reyna Abhyankar, Qi Qi, and Yiying Zhang. OSWorld-Human: Benchmarking the efficiency of computer-use agents. In A. Chowdhery and Z. Jia, editors, *Proceedings of Machine Learning and Systems*, volume 8, pages 482–494. MLSys, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/file/5edb57c05c81d04beb716ef1d542fe9e-Paper-Conference.pdf.

Jinyuan Chen, Jiuchen Shi, Quan Chen, and Minyi Guo. Kairos: Low-latency multi-agent serving with shared LLMs and excessive loads in the public cloud. *arXiv preprint arXiv:2508.06948*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tao Feng, Haozhen Zhang, Zijie Lei, Peixuan Han, and Jiaxuan You. GraphPlanner: Graph memory-augmented agentic routing for multi-agent LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ZdGB7MNQDT.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. Language model compression with weighted low-rank factorization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uPv9Y3gmAI5.

Ahasan Kabir, Jiaqi Xue, Mengxin Zheng, and Qian Lou. HW-Router: Hardware-aware routing for scalable multi-LLM serving. In *Proceedings of the 63rd ACM/IEEE Design Automation Conference*, DAC '26, New York, NY, USA, 2026a. Association for Computing Machinery. ISBN 9798400722547. doi: 10.1145/3770743.3804427. URL https://doi.org/10.1145/3770743.3804427.

Ahasan Kabir, Jiaqi Xue, Mengxin Zheng, and Qian Lou. INFRAMIND: Infrastructure-aware multi-agent orchestration, 2026b. URL https://arxiv.org/abs/2606.11440.

Hao Kang, Qingru Zhang, Han Cai, Weiyuan Xu, Tushar Krishna, Yilun Du, and Tsachy Weissman. Win fast or lose slow: Balancing speed and accuracy in latency-sensitive decisions of LLMs. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 150862–150884. Curran Associates, Inc., 2025. doi:

10.52202/085713-5044. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/ddaec864ba433e8889ab08dcf5c26e55-Paper-Conference.pdf.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 51991–52008. Curran Associates, Inc., 2023. doi: 10.52202/075280-2264. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a3621ee907def47c1b952ade25c67698-Paper-Conference.pdf.

Qian Lou, Feng Guo, Minje Kim, Lantao Liu, and Lei Jiang. AutoQ: Automated kernel-wise neural network quantization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygfnn4twS.

Qian Lou, Yen-Chang Hsu, Burak Uzkent, Ting Hua, Yilin Shen, and Hongxia Jin. Lite-MDETR: A lightweight multi-modal detector. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12206–12215, June 2022a.

Qian Lou, Ting Hua, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. DictFormer: Tiny transformer with shared dictionary. In *International Conference on Learning Representations*, 2022b. URL https://openreview.net/forum?id=GWQWAeE9EpB.

OpenAI. GPT-4o mini: advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, July 2024. Accessed: 2026-01-06.

Rakibul Hasan Rajib, Mengxin Zheng, and Qian Lou. Learning what to retain: Gated-memory routing for efficient collaboration in multi-agent LLM systems. In Hung-Yi Lee, Sunipa Dev, Junyi Jessy Li, and Desmond Elliott, editors, *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, pages 39594–39611, Budapest, Hungary, October 2026a. Association for Computational Linguistics. ISBN 979-8-89176-459-0. doi: 10.18653/2026.emnlp-main.1735. URL https://aclanthology.org/2026.emnlp-main.1735/.

Rakibul Hasan Rajib, Mengxin Zheng, and Qian Lou. AgentServeSim: Serving-system simulation and policy search for LLM agent programs, 2026b. URL https://arxiv.org/abs/2606.09613.

Xi Shi, Mengxin Zheng, and Qian Lou. KVShareArena: KV-cache reuse across contexts and model checkpoints, 2026. URL https://arxiv.org/abs/2609.10266.

Song Wang, Zhen Tan, Zihan Chen, Shuang Zhou, Tianlong Chen, and Jundong Li. AnyMAC: Cascading flexible multi-agent collaboration via next-agent prediction. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 11555–11567, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.584. URL https://aclanthology.org/2025.emnlp-main.584/.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yingxu Wang, Siwei Liu, Jinyuan Fang, and Zaiqiao Meng. EvoAgentX: An automated framework for evolving agentic workflows. In Ivan Habernal, Peter Schulam, and Jörg Tiedemann, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 643–655, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-334-0. doi: 10.18653/v1/2025.emnlp-demos.47. URL https://aclanthology.org/2025.emnlp-demos.47/.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet,

J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 95266–95290. Curran Associates, Inc., 2024. doi: 10.52202/079017-3018. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ad236edc564f3e3156e1b2feafb99a24-Paper-Datasets_and_Benchmarks_Track.pdf.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. AgentDropout: Dynamic agent elimination for token-efficient and high-performance LLM-based multi-agent collaboration. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24013–24035, Vienna, Austria, July 2025c. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1170. URL https://aclanthology.org/2025.acl-long.1170/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

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 conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Jiaqi Xue, Qian Lou, Jiarong Xing, and Heng Huang. R2-Router: A new paradigm for LLM routing with reasoning. In Tong Zhang, Miroslav Dudik, Martin Jaggi, Alekh Agarwal, Sharon Li, Dale Schuurmans, Jerry Zhu, Felix Berkenkamp, Hanze Dong, and Alberto Bietti, editors, *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 143255–143268. PMLR, 06–11 Jul 2026. URL https://proceedings.mlr.press/v306/xue26h.html.

Yu Yao, Yiliao (Lia) Song, Yian Xie, Mengdan Fan, Mingyu Guo, and Tongliang Liu. Can dependencies induced by LLM-agent workflows be trusted? In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 17431–17469. Curran Associates, Inc., 2025. doi: 10.52202/085713-0587. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/19206a6ed5ed0aaeed440448dfc5cf7e-Paper-Conference.pdf.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 75834–75852. PMLR, 13–19 Jul 2025a. URL https://proceedings.mlr.press/v267/zhang25bi.html.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for LLM-based multi-agent systems. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=LkzuPorQ5L.

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 Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 76678–76692. PMLR, 13–19 Jul 2025c. 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 *The Thirteenth International Conference on Learning Representations*, 2025d. URL https://openreview.net/forum?id=z5uVAKwmjf.

Changsheng Zhao, Ting Hua, Yilin Shen, Qian Lou, and Hongxia Jin. Automatic mixed-precision quantization search of BERT. In Prof. Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 3427–3433. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/472. URL https://doi.org/10.24963/ijcai.2021/472. Main Track.

Jiaru Zou, Ruizhong Qiu, Gaotang Li, Xiyuan Yang, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, James Zou, Mengdi Wang, and Ling Yang. Latent collaboration in multi-agent systems. In Tong Zhang, Miroslav Dudik, Martin Jaggi, Alekh Agarwal, Sharon Li, Dale Schuurmans, Jerry Zhu, Felix Berkenkamp, Hanze Dong, and Alberto Bietti, editors, *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 168813–168847. PMLR, 06–11 Jul 2026. URL https://proceedings.mlr.press/v306/zou26k.html.

A Limitations

LAMaS reduces orchestration latency; however, it does not mitigate broader risks inherited from the underlying LLMs and associated tools, including unsafe outputs, hallucinations, and social biases.

B Experimental Setup Details

This section collects additional implementation and dataset details used for reproducibility.

B.1 Operator Set

Each operator corresponds to a predefined reasoning or execution primitive used to construct execution graphs. We briefly summarize their functionality below for completeness.

- **Generate.** A basic generator that directly produces text or code without additional reasoning or post-processing. It invokes the LLM once and is primarily used for simple generation tasks.
- **GenerateCoT.** A chain-of-thought generator that prompts the LLM to perform step-by-step reasoning. For mathematical tasks (e.g., MATH, GSM8K), it includes explicit reasoning exemplars, while for code tasks (e.g., HumanEval), it uses lightweight reasoning prompts. This operator invokes the LLM once.
- **MultiGenerateCoT.** A diversity-oriented CoT generator that produces multiple candidate solutions in parallel. It generates three independent chain-of-thought solutions, yielding a set of candidate responses for downstream aggregation.
- **ScEnsemble.** A self-consistency ensemble operator that selects the most consistent answer from multiple candidate solutions. All candidates are formatted as discrete options, and the LLM is prompted to select the most consistent one, following the self-consistency principle.
- **SelfRefine.** A refinement operator that analyzes an existing solution to identify errors or suboptimal reasoning and generates an improved version. This operator invokes the LLM once.

Task-Specific Operators. We also employ task-specific operators for different benchmarks.

- **CustomCodeGenerate** (HumanEval). A lightweight code generator that produces candidate code solutions without execution or testing.
- **Test** (HumanEval). A test-driven refinement operator that executes generated code and iteratively improves it based on failure feedback. Upon failure, the operator generates a revised solution using reflective prompts and retries up to three times.
- **Programmer** (MATH/GSM8K). A code execution operator that generates Python programs, executes them in an isolated environment, and iteratively refines the code based on execution feedback.

B.2 Additional Implementation Details

We list the remaining training hyperparameters not detailed in Section 4.1. The cost weight is $\lambda_c = 3.0$, the dual learning rate is $\eta = 0.01$, the batch size is 4, and the orchestrator is trained for 4 repetitions over the training set. These values follow the supernet implementation conventions of MaAS [Zhang et al., 2025a].

Queries are encoded by a frozen `all-MiniLM-L6-v2` SentenceTransformer; the resulting embedding is shared between the orchestrator's policy network (Section 3.2) and the controller's Compress$(\cdot)$ projection (Section 3.6), and is not finetuned during training.

In the main experiments, we use `gpt-4o-mini-0718` via the OpenAI API with temperature set to 1. Appendix E instead evaluates `qwen3.5-flash` (closed-source) via OpenRouter with `Alibaba Cloud Int.` selected as the provider, and `qwen3.5-9b` (open-weight) served locally through vLLM; the same temperature setting is used there as well.

Training the orchestrator and execution controller requires modest local compute but does not finetune the underlying LLM backbone. All training runs were executed on a Slurm cluster using a single NVIDIA H100 80GB GPU per run. The local `qwen3.5-9b` evaluation uses vLLM served on two NVIDIA B200 GPUs.

B.3 Dataset Split Statistics

We evaluate on four publicly available benchmarks: HumanEval [Chen et al., 2021], GSM8K [Cobbe et al., 2021], MATH [Hendrycks et al., 2021], and MMLU-Pro [Wang et al., 2024]. Following common practice in workflow-automation work [Zhang et al., 2025a,d], each benchmark is partitioned into training and test sets with a train:test ratio of 1:4. For MATH, we use a 605-problem subset of level-5 problems spanning four representative categories: Combinatorics & Probability, Number Theory, Pre-algebra, and Pre-calculus. For MMLU-Pro, we construct a stratified split from the official test set (seed 42), sampling proportionally per category with no overlap between train and test. Statistics are summarized in Table 4.

Table 4: Dataset statistics.

Domain	Dataset	#Train	#Test	Metric
Code Generation	HumanEval	33	131	pass@1
Math Reasoning	GSM8K	264	1055	Accuracy
	MATH	119	486	Accuracy
Knowledge Reasoning	MMLU-Pro	125	500	Accuracy

B.4 Baseline Setups

We provide the configurations for each baseline method below.

1. **Generate.** A single-pass direct-answer baseline that prompts the LLM once with the query and uses its output as the final answer, with no chain-of-thought scaffolding or post-processing.
2. **CoT.** Chain-of-Thought prompting [Wei et al., 2022] elicits step-by-step reasoning before the final answer; we use a standard zero-shot CoT prompt template.
3. **SC.** Self-Consistency [Wang et al., 2023]: to enhance robustness, we aggregate five CoT-generated solutions through an ensemble step over the candidate answers.
4. **GDesigner.** We use the official implementation of Zhang et al. [2025c], which learns a graph neural network over operator embeddings to construct communication topologies. We retain the original training procedure and operator set defined in the released codebase, replacing only the LLM backbone with `gpt-4o-mini-0718` for fair comparison.
5. **AgentDropout.** We follow the released implementation of Wang et al. [2025c], which dynamically prunes redundant agents and communication edges during execution. The dropout schedule and per-agent role assignments use the defaults from the official codebase.
6. **MaAS.** We use the agentic-supernet implementation of Zhang et al. [2025a], training the orchestrator with the same supernet depth ($L = 4$), operator set, and policy-gradient procedure as released.

The original cost-only reward is used unchanged so that MaAS represents the cost-aware—but not latency-aware—learning baseline.

All MAS baselines are trained and evaluated under the same train/test partition (Appendix B.3), the same backbone LLM (`gpt-4o-mini-0718`), and the same per-query latency/cost measurement protocol described in Section 4.1.

C Additional Supporting Analyses

This section provides supplementary analyses supporting the main-paper interpretation of LAMaS.

C.1 Critical-Path Tokens as a Latency Proxy

The critical-path-aware credit assignment in Section 3.4 is motivated by the expectation that CP Tokens should co-vary with end-to-end wall-clock latency. To verify this empirically, we form one $(\text{CP Tokens}, \text{latency})$ pair for each MAS-method/benchmark cell in Table 1 and compute the pooled correlation across all 4 MAS methods $\times$ 4 benchmarks ($n = 16$). Both Pearson and Spearman correlations indicate a strong positive association, providing complementary evidence that reductions in CP Tokens align with reductions in end-to-end latency.

Table 5: Correlation between CP Tokens and measured wall-clock latency across MAS-method/benchmark cells in Table 1.

Setting	Pearson r	Spearman ρ
MAS methods $\times$ 4 benchmarks	0.995	0.985

C.2 Comparison with Oracle Controller

To probe how close the learned execution controller is to an optimal stopping rule, we compare it against an oracle that has access to the true control advantage $\Delta_k = R_k - R_N$ at each decision point and stops iff $\Delta_k > 0$. We evaluate on HumanEval and MMLU-Pro from two complementary angles: decision-level agreement with the oracle (Table 6), and end-to-end performance when the learned controller is replaced by either the oracle or no controller (Table 7).

Decision-level agreement. Table 6 reports decision accuracy together with precision, recall, and F1 for the positive (stop) class, treating the oracle's $\Delta_k > 0$ rule as ground truth. Across both benchmarks, the learned controller agrees with oracle decisions in 70.8–73.5% of cases. Precision exceeds recall on both benchmarks, indicating that the learned controller is conservative: it tends to continue execution when the oracle's signal is borderline rather than making an over-aggressive elimination decision.

Table 6: Decision-level agreement between the learned execution controller and the oracle stopping rule on HumanEval and MMLU-Pro. The positive class is the "stop" decision.

Benchmark	Decision Acc (%) ↑	Precision ↑	Recall ↑	F1 ↑
HumanEval	73.5	0.72	0.66	0.69
MMLU-Pro	70.8	0.69	0.63	0.66

End-to-end performance. Table 7 compares three controller settings: no controller, the learned controller, and the oracle. The learned controller closes most of the gap between no-controller execution and the oracle: on HumanEval it trails the oracle by 2.16 s in latency and 0.25 pp in accuracy, and on MMLU-Pro by 1.74 s and 0.46 pp. This indicates that the learned controller approaches oracle-level execution control despite operating only on the partial-execution features available at runtime.

C.3 Full Training and Inference Procedure

Table 7: End-to-end accuracy and latency under different controllers on HumanEval and MMLU-Pro. The "Learned" rows correspond to the LAMaS configuration in Table 2, and "No controller" rows correspond to the *w/o controller* ablation.

Benchmark	Controller	Acc (%) ↑	Latency (s) ↓
HumanEval	No controller	95.17	24.47
	Learned	95.42	21.00
	Oracle	**95.67**	**18.84**
MMLU-Pro	No controller	66.00	17.94
	Learned	66.27	14.81
	Oracle	**66.73**	**13.07**

Algorithm 1 LAMaS: Latency-Aware MAS Orchestration

Require: Orchestrator π_θ, operator set $\mathcal{O}$, training queries $\mathcal{D}$, accuracy floor S_0
Require: Hyperparameters: λ_t, λ_c, dual learning rate η, threshold τ
// Phase 1: Train orchestrator
Initialize dual variable $\lambda_S \leftarrow 1.0$
for each training iteration **do**
  Sample a batch of queries $\{x_i\}$ from $\mathcal{D}$
  for each query x_i **do**
    Construct execution graph $\mathcal{G}_i$ via π_θ
    Execute $\mathcal{G}_i$; observe S_i, C_i, T_i, per-operator times $\{t(o)\}$
    Compute $\ell(o)$ for all operators; compute $R(o)$ via Eq. 8
  end for
  Update θ via policy gradient (Eq. 9)
  $\lambda_S \leftarrow \max(0,\ \lambda_S + \eta(1 - \bar{S}/S_0))$
end for
// Phase 2: Train execution controller
Freeze π_θ; collect full execution traces on $\mathcal{D}$
for each trace **do**
  for each decision point $k = 1, \ldots, N$ **do**
    Compute $\Delta_k = R_k - R_N$ (Eq. 10)
    Record training sample: (f_k, Δ_k)
  end for
end for
Train Controller$_\phi$ via MSE: $\min_\phi \sum(\hat{\Delta}_k - \Delta_k)^2$
Compute $\sigma_\Delta = \text{std}(\{\Delta_k \mid \Delta_k > 0\})$
// Inference
Construct $\mathcal{G}$ via π_θ; launch operators
for each operator completion k **do**
  Compute $p(\text{select current})$ from Controller$_\phi(f_k)$, sample a Bernoulli decision; on a positive draw, eliminate the following unlaunched operators.
end for
return Final answer

C.4 Compatibility with Unseen Operators

To probe whether the learned policy generalizes beyond the fixed training-time operator set, we insert an unseen operator (**Debate**) at inference time on HumanEval and inspect the orchestrator's average selection probabilities. Figure 4 reports the average probability mass over the seven non-halt operators shown. The unseen **Debate** operator receives 14.8% probability mass, comparable to seen operators such as **Generate** (14.9%), **ScEnsemble** (15.1%), and **Test** (15.5%). This suggests that the learned policy can still reasonably select an unseen operator during architecture sampling, indicating some inductive flexibility beyond the exact operator set seen during training.

D Sensitivity Analyses

This section reports the sensitivity of LAMaS to the main hyperparameters that control latency pressure, graph structure, and the accuracy floor.

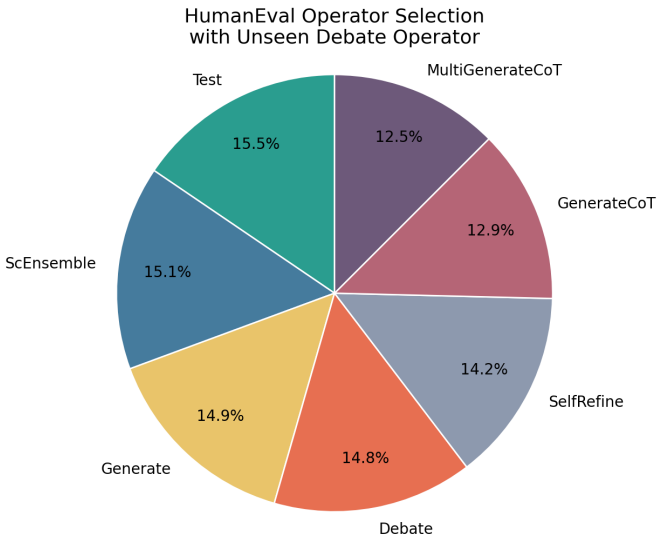

Figure 4: Average operator selection probabilities on HumanEval after inserting an unseen operator, **Debate**, at inference time. The unseen operator receives non-trivial probability mass comparable to several seen operators, indicating that the learned policy can still reasonably select operators not exposed during training.

D.1 Parameter Sensitivity Analysis

We analyze the sensitivity of LAMaS to four key hyperparameters: the latency penalty weight λ_t, the activation threshold τ, the number of supernet layers L, and the accuracy floor S_0. We report results on HumanEval and MMLU-Pro, with the default configuration (λ_t=0.005, τ=0.3, L=4) marked by ♦.

Table 8: Sensitivity to the latency penalty weight λ_t. Increasing λ_t consistently reduces latency, with mild accuracy variation across the sweep.

Dataset	λ_t	Acc (%) ↑	Latency (s) ↓
HumanEval	0	95.67	35.62
	0.001	**95.93**	23.50
	0.005 ♦	95.42	21.00
	0.01	94.91	13.91
	0.05	93.89	**11.29**
MMLU-Pro	0	66.33	22.46
	0.001	**67.13**	16.41
	0.005 ♦	66.27	14.81
	0.01	66.13	14.11
	0.05	65.27	**11.38**

Table 9: Sensitivity to the activation threshold τ. A higher threshold activates more operators per step, increasing accuracy and latency. A lower threshold produces leaner graphs with lower latency but reduced accuracy.

Dataset	τ	Acc (%) ↑	Latency (s) ↓
HumanEval	0.1	93.89	**13.57**
	0.2	94.91	15.35
	0.3 ♦	95.42	21.00
	0.4	**95.67**	22.80
MMLU-Pro	0.1	65.13	**13.42**
	0.2	65.93	14.23
	0.3 ♦	66.27	14.81
	0.4	**66.67**	15.62

Table 10: Sensitivity to the number of supernet layers L. More layers provide greater reasoning depth and slightly higher accuracy, but increase latency due to deeper critical paths.

Dataset	L	Acc (%) ↑	Latency (s) ↓
HumanEval	2	94.15	**12.85**
	4 ♦	95.42	21.00
	6	95.67	23.50
	8	**95.93**	25.54
MMLU-Pro	2	64.93	**12.81**
	4 ♦	66.27	14.81
	6	66.47	16.56
	8	**66.53**	17.43

Latency weight λ_t. Table 8 shows that increasing λ_t monotonically reduces latency, while accuracy varies only mildly across the sweep. On HumanEval, raising λ_t from 0 to 0.05 reduces latency from 35.62s to 11.29s with at most 2.0% accuracy change; the trend is consistent on MMLU-Pro. The λ_t=0 setting (no latency objective during training) leaves accuracy near the LAMaS level but yields the largest latency, complementing the corresponding row in the ablation study (Table 2). Overall, λ_t provides smooth and predictable control over latency while keeping accuracy broadly stable.

Activation threshold τ. Table 9 reveals that the threshold controls the width of the execution graph. A lower threshold (τ=0.1) produces lean graphs with fewer operators per step, yielding the lowest latency but reduced accuracy. A higher threshold (τ=0.4) activates more operators, improving accuracy through greater exploration together with increased latency. The default τ=0.3 provides a balanced setting across both benchmarks.

Number of layers L. Table 10 shows that increasing the number of supernet layers provides greater reasoning depth. On HumanEval, accuracy improves from 94.15% (L=2) to 95.93% (L=8), but latency increases from 12.85s to 25.54s due to the deeper critical path. The marginal accuracy gain diminishes beyond L=4, while latency continues to grow, justifying the default choice of L=4.

Accuracy floor S_0. Unlike λ_t, τ, and L, which control the latency penalty and graph structure, the accuracy floor S_0 sets the target accuracy level used during training. Table 11 reports results across $S_0 \in \{0.95, 1.00, 1.02, 1.05\} \cdot S_{\text{ref}}$, where S_{ref} denotes the orchestrator's reference accuracy and the default deployment ($S_0 = S_{\text{ref}}$) is marked by ♦. The trend is monotonic on both benchmarks: on HumanEval, sweeping S_0 from $0.95 \cdot S_{\text{ref}}$ to $1.05 \cdot S_{\text{ref}}$ moves accuracy from 94.66% to 96.18% while latency rises from 17.85s to 23.54s; the same direction holds on MMLU-Pro (65.40%→67.13%, 12.60s→16.51s).

Table 11: Sensitivity to the accuracy floor S_0. We vary S_0 as a fraction of the architecture's reference accuracy S_{ref} ($S_0 = S_{\text{ref}}$ is the default, marked by ♦).

Dataset	S_0/S_{ref}	Acc (%) ↑	Latency (s) ↓
HumanEval	0.95	94.66	**17.85**
	1.00 ♦	95.42	21.00
	1.02	95.67	22.00
	1.05	**96.18**	23.54
MMLU-Pro	0.95	65.40	**12.60**
	1.00 ♦	66.27	14.81
	1.02	66.67	15.46
	1.05	**67.13**	16.51

E Generalizability Across Backbones and Serving Setups

To assess whether LAMaS's latency advantage transfers across different LLM backbones and serving setups, we replicate the comparison on two additional configurations: the closed-source flash variant `qwen3.5-flash` served through OpenRouter, and the open-weight `qwen3.5-9b` served locally with vLLM. Both are evaluated on HumanEval and MMLU-Pro against the strongest learning-based

MAS baseline (MaAS) and LAMaS. As shown in Table 12, LAMaS reduces latency by about 60% on HumanEval and about 43% on MMLU-Pro across the two settings, while matching or slightly exceeding MaAS in accuracy. This confirms that LAMaS's mechanisms transfer across distinct backbones and serving setups.

Table 12: Generalizability across Qwen backbones and serving setups: closed-source `qwen3.5-flash` served through OpenRouter and open-weight `qwen3.5-9b` served locally with vLLM. Latency is end-to-end wall-clock time per query (s). Best mean among MAS methods per column per dataset is in **bold**.

Dataset	Backbone	Method	Acc (%) ↑	Latency (s) ↓
HumanEval	`qwen3.5-flash`	MaAS	94.91	100.87
		LAMaS	**96.95**	**38.93**
	`qwen3.5-9b`	MaAS	95.17	105.18
		LAMaS	**95.93**	**42.18**
MMLU-Pro	`qwen3.5-flash`	MaAS	82.40	174.60
		LAMaS	**82.93**	**98.04**
	`qwen3.5-9b`	MaAS	71.20	178.62
		LAMaS	**72.93**	**102.07**

F Additional Baseline Comparisons

This section reports two additional comparisons: single-agent agentic baselines, and naive latency-cutting rules applied to an existing MAS baseline.

F.1 Single-Agent Agentic Baselines

The non-MAS baselines in Table 1 are prompting and sampling methods. We additionally evaluate three single-agent agentic baselines, equipping each task with the tool or self-correction mechanism it naturally supports. **Tool-augmented CoT** on GSM8K writes and executes code and answers from the execution result (the **Programmer** operator). **Test-time repair** on HumanEval runs the generated program and rewrites it from the failure feedback for up to three iterations (the **Test** operator). **Self-Refine** on MMLU-Pro critiques and revises its own answer in one round (the **SelfRefine** operator). All three reuse the operator implementations and prompts of Appendix B.1 and are evaluated under the protocol of Section 4.1.

Table 13: Comparison with single-agent agentic baselines. LAMaS rows are taken from Table 1. Cost is in units of $\times 10^{-4}$ USD; latency is end-to-end wall-clock time per query (in seconds).

Dataset	Method	Acc (%) ↑	Cost ($\times 10^{-4}$) ↓	Latency (s) ↓
GSM8K	Tool-augmented CoT	89.79	11.38	18.27
	LAMaS	93.65	7.71	11.73
HumanEval	Test-time repair	86.51	6.35	15.43
	LAMaS	95.42	12.92	21.00
MMLU-Pro	Self-Refine	57.13	5.51	15.19
	LAMaS	66.27	10.93	14.81

As shown in Table 13, LAMaS is 3.86 pp more accurate than Tool-augmented CoT on GSM8K, 8.91 pp more accurate than Test-time repair on HumanEval, and 9.14 pp more accurate than Self-Refine on MMLU-Pro; on GSM8K it is also cheaper and faster. These baselines and SC (Table 1) provide complementary controls: the agentic baselines test tool use and self-correction, and SC tests additional sampling and ensembling. SC incurs higher cost than LAMaS on all four benchmarks yet remains 9.16 and 7.60 pp less accurate on HumanEval and MMLU-Pro, respectively. Since the operator implementations are shared, what differs between these baselines and LAMaS is which operators run and how they are arranged.

F.2 Naive Latency-Cutting Rules on an Existing MAS Baseline

The ablation in Table 2 covers naive latency optimization at training time (*naive lat. only*). To examine how an existing MAS baseline behaves under simple latency-cutting rules at inference time, we evaluate GDesigner under two rules that require no training. **Consensus-2 stop** terminates all not-yet-launched agents as soon as any two completed agents produce agreeing answers under the agreement rule of Section 3.6. **Half-agents** randomly removes $\lfloor n/2 \rfloor$ of the n agents while retaining the final aggregation step. Both rules are evaluated on GSM8K and MMLU-Pro under the protocol of Section 4.1.

Table 14: GDesigner under two naive inference-time latency-cutting rules. The GDesigner and GDesigner+LAMaS rows are taken from Tables 1 and 3. Cost is in units of $\times 10^{-4}$ USD; latency is end-to-end wall-clock time per query (in seconds).

Dataset	Method	Acc (%) ↑	Cost ($\times 10^{-4}$) ↓	Latency (s) ↓
GSM8K	GDesigner	93.30	17.86	32.31
	GDesigner + consensus-2 stop	92.61	13.97	26.92
	GDesigner + half-agents	91.37	11.24	21.37
	GDesigner+LAMaS	93.27	12.08	20.55
MMLU-Pro	GDesigner	64.27	25.03	46.66
	GDesigner + consensus-2 stop	62.93	19.02	39.48
	GDesigner + half-agents	61.13	15.48	29.34
	GDesigner+LAMaS	64.33	17.03	28.27

Table 14 shows that neither rule matches GDesigner+LAMaS. Consensus-2 stop is dominated on accuracy, cost, and latency, while half-agents trades accuracy for cost: at nearly the same latency, it is cheaper but 1.90 and 3.20 pp less accurate on GSM8K and MMLU-Pro, respectively. Naive latency cutting, whether at training time (Table 2) or at inference time (Table 14), sacrifices accuracy; among these variants, only the LAMaS pipeline reduces latency while preserving it.

NeurIPS Paper Checklist

1. **Claims**
   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?
   Answer: [Yes]
   Justification: The abstract and Section 1 state the paper's claims: (i) the constrained-optimization formulation with critical-path-aware credit assignment (Section 3), (ii) the inference-time execution-time controller (Section 3.6), (iii) empirical evaluation across four benchmarks plus transfer to other learning-based MAS architectures (Section 4), and the platform interface described in Section 5. The reported quantitative claims (e.g., over 50% latency reduction, competitive accuracy) match the experimental tables.
   Guidelines:
   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.
2. **Limitations**
   Question: Does the paper discuss the limitations of the work performed by the authors?
   Answer: [Yes]
   Justification: The paper includes a dedicated Limitations section that notes LAMaS targets orchestration efficiency and does not mitigate broader risks inherited from the underlying LLMs and tools (unsafe outputs, hallucinations, social biases). Methodological scope is also discussed throughout: the framework targets learning-based orchestrators, the experiments use a single closed-source backbone (`gpt-4o-mini`; additional Qwen3.5-Flash and Qwen3.5-9B backbones are reported in the appendix), and the operator set follows prior work.
   Guidelines:
   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.
3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present formal theorems. The Lagrangian relaxation and dual update rules in Section 3 follow standard constrained-optimization machinery and are stated as derivations rather than as theorems requiring proof.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Algorithm 1 specifies the full training and inference procedure. Section 4.1 and Appendix B.2 report the LLM backbone (`gpt-4o-mini-0718`, temperature 1), benchmark splits (following MaAS), the main hyperparameters (λ_t, τ, supernet depth L), and the S_0 setup; Section 3.6 describes the controller's training procedure; the remaining training hyperparameters (λ_c, η) are listed in the appendix. Datasets (GSM8K, HumanEval, MATH, MMLU-Pro) are publicly available with citations.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code is released at `https://github.com/loopleap-lab/LAMaS`, covering the orchestrator and execution-time controller along with configuration files and instructions for use. All four datasets (GSM8K, HumanEval, MATH, MMLU-Pro) are publicly available and cited.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 ("Experimental Setup") reports the benchmarks and splits, baselines, evaluation metrics (accuracy, cost, latency), backbone LLM, and all hyperparameter settings. Sensitivity analyses for λ_t, τ, L, and S_0 are provided in Appendix D to show how each setting was chosen and how robust the results are to these choices.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Each cell in Table 1 (the main experiment) reports the mean $\pm$ 1-sigma standard deviation across 3 independent runs of the full evaluation pipeline. The variability captured is run-to-run variation arising from LLM sampling stochasticity and the resulting orchestration trajectory differences. Standard deviations are computed directly from the 3 runs (no parametric assumption). The ablation, sensitivity, and transferability tables report the corresponding 3-run means without explicit error bars for space reasons.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 4.1 and Appendix B.2 report the backbone access setup and the local training hardware. Specifically, the orchestrator and execution controller are trained on a Slurm cluster using a single NVIDIA H100 80GB GPU per run, while the underlying LLM backbones are accessed by API rather than finetuned locally. The paper also reports the per-query latency and monetary cost of evaluation, which together characterize the dominant runtime resources for the API-based experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics. The work uses publicly available reasoning benchmarks and a closed-source LLM accessed via standard APIs; it does not involve human subjects, sensitive data, or content with privacy concerns.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The Limitations section notes that LAMaS targets orchestration efficiency and does not mitigate risks inherited from underlying LLMs (unsafe outputs, hallucinations, social biases). On the positive side, lower inference latency can make MAS more practical for interactive and latency-sensitive applications.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We do not release pretrained generative models or scraped datasets. The released artifacts are the orchestrator-and-controller training code together with thin wrappers around publicly available reasoning benchmarks; these do not pose meaningful misuse risk on their own.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The datasets used in the paper are publicly available and properly cited.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The current submission does not release new datasets, pretrained model weights, or other standalone assets that require accompanying documentation. If code is released in a later version, it will be documented accordingly.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve research with human subjects, so IRB approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for

writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are used as the underlying agents within the multi-agent system being orchestrated. The specific backbones (`gpt-4o-mini-0718`, with Qwen3.5-Flash and Qwen3.5-9B as secondary backbones in the appendix), the temperature setting, and the operator set that wraps each LLM invocation are documented in Section 4.1 and Appendices B.1 and B.2. The methodological contribution—the orchestrator, the constrained Lagrangian formulation, the critical-path-aware credit assignment, and the inference-time execution controller—is independent of any specific LLM and operates on top of these models as a substrate.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.