NOT EVERY AGENT NEEDS THE STRONGEST MODEL: UNIFIED PROMPT–MODEL OPTIMIZATION FOR HETEROGENEOUS LLM-BASED MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM)-based multi-agent systems (MAS) rely on role-specific prompts and execution models to solve complex tasks collaboratively. However, prompt optimization typically keeps model assignments fixed, leaving differences in role requirements and their associated inference costs outside the search. We present **UNIMASO**, a **Uni**fied framework for heterogeneous **M**ulti-**A**gent **S**ystem **O**ptimization through the joint adaptation of prompts and model assignments. UniMASO evaluates candidate updates through local, downstream, and global feedback, using cost-aware Pareto search to retain useful quality–cost trade-offs. Topology-aware co-evolution then coordinates these updates according to agent roles and dependencies. Cross-level trajectory memory further supplies historical successes and failures to guide subsequent optimization. Experiments on five benchmarks and five execution topologies show that UniMASO achieves competitive task performance at substantially lower deployment cost than the strongest homogeneous configurations. Under centralized execution, it improves the average score from 69.79 to 70.40 while reducing token-weighted deployment cost by 85.2% relative to DeepSeek-V4-Pro with MASPO. These results support joint prompt–model adaptation as an effective approach to efficient multi-agent collaboration. The code is available at `https://anonymous.4open.science/r/UniMASO-864D`.

1 INTRODUCTION

LLM-based multi-agent systems (MAS) decompose complex tasks through role specialization and communication (Li et al., 2023; Wu et al., 2024; Chen et al., 2024b; Hong et al., 2024; Zhuge et al., 2024). Each agent's prompt shapes its behavior and the information passed downstream, while its execution model determines available capabilities and inference cost. System quality therefore depends on how both choices align across the communication topology (Table 1).

Table 1: Comparison of prompt and model optimization paradigms.

Method	Prompt	Model
MIPRO (Opsahl-Ong et al., 2024)	✓	✗
GPTSwarm (Zhuge et al., 2024)	✓	✗
MASS (Zhou et al., 2026)	✓	✗
MASPOB (Hong et al., 2026)	✓	✗
MASPO (Wang et al., 2026)	✓	✗
MasRouter (Yue et al., 2025)	✗	✓
Hetero-Designer (Zhang et al., 2026b)	✗	✓
UNIMASO (Ours)	✓	✓

Automatic prompt optimization refines instructions from task feedback, and recent MAS methods coordinate role prompts (Yang et al., 2024; Fernando et al., 2024; Opsahl-Ong et al., 2024; Cheng et al., 2024; Zhou et al., 2026; Zhang et al., 2026a; Wang et al., 2026; Hong et al., 2026); they generally fix execution models. Conversely, model routing and heterogeneous agent design assign models by task or role (Chen et al., 2024a; Ong et al., 2025; Chen et al., 2025; Yue et al., 2025; Zhang et al., 2026b), but typically fix prompts, leaving behavior outside model search. Fixed model assignments create *role–capability misalignment*: they can waste strong models on routine roles or bottleneck demanding roles with lightweight models, while prompt edits cannot supply missing capabilities.

Changes to either choice propagate through the agent graph, altering downstream context, end-to-end quality, and cost; independent search can favor locally attractive but system-level poor configurations. This coupling motivates joint prompt–model search across the full execution topology.

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

To address this gap, we introduce **UniMASO**, which jointly optimizes agent prompts and model assignments in heterogeneous MAS (Figure 1). It evaluates candidate updates through local, downstream, and global feedback, then applies cost-aware Pareto search to retain quality–cost trade-offs. Topology-aware co-evolution coordinates updates across dependent agents, while cross-level trajectory memory guides subsequent search. Across five benchmarks and five execution topologies, we assess quality–cost trade-offs, topology generalization, and the contributions of trajectory memory and search controls. Under centralized execution, UniMASO raises the average score from 69.79 to 70.40 while reducing token-weighted deployment cost by 85.2% relative to DeepSeek-V4-Pro with MASPO.

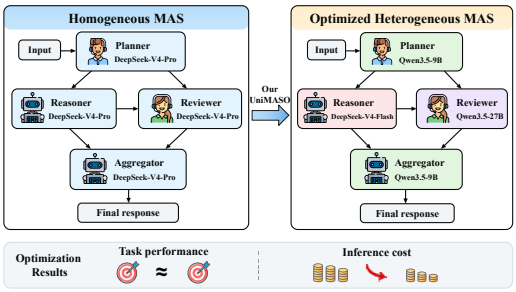

Figure 1: Homogeneous vs. Heterogeneous MAS.

Our contributions are summarized as follows:

- We formulate heterogeneous MAS optimization over prompt–model configurations, making role–capability misalignment explicit.
- We develop cost- and topology-aware search that couples local multi-granularity evaluation with system-level co-evolution.
- We introduce cross-level trajectory memory that turns local feedback, global outcomes, and cost observations into experience for later rounds.

2 Related Work

2.1 Multi-Agent Systems and Prompt Optimization

LLM-based multi-agent systems coordinate model calls through specialized roles and message exchange. CAMEL, AutoGen, AgentVerse, and MetaGPT establish role-based workflows (Li et al., 2023; Wu et al., 2024; Chen et al., 2024b; Hong et al., 2024), while GPTSwarm and MASS automate system construction by optimizing agent graphs or co-evolving prompts and topologies (Zhuge et al., 2024; Zhou et al., 2026). Prompt optimizers search natural-language instructions through candidate scores, evolutionary search, execution traces, or language feedback (Yang et al., 2024; Fernando et al., 2024; Cheng et al., 2024; Agrawal et al., 2026). MIPRO jointly optimizes instructions and demonstrations in modular programs (Opsahl-Ong et al., 2024). Because upstream outputs shape successor context, local prompt gains need not improve final responses. MAPRO coordinates node-wise candidates with probabilistic inference (Zhang et al., 2026a), whereas MASPO evaluates local, downstream, and global effects through topology-guided search (Wang et al., 2026). MASPOB allocates optimization budgets using graph representations and bandit search (Hong et al., 2026). These methods keep node-level execution models fixed. UniMASO jointly searches prompts and model assignments, using cross-round trajectories to balance system utility and inference cost.

2.2 Heterogeneous Model Assignment

Model capability and invocation cost motivate cost-aware model selection. FrugalGPT reduces inference cost through learned cascades, while RouteLLM routes queries between models using preference data (Chen et al., 2024a; Ong et al., 2025). Compound language-model systems optimize model combinations under performance and budget constraints (Chen et al., 2025). In multi-agent settings, MasRouter selects execution models based on task and agent characteristics, while HeteroDesigner jointly assigns roles, models, and collaboration structures (Yue et al., 2025; Zhang et al., 2026b). These methods route models under fixed prompts or configure each query independently, decoupling model choice from the evolving prompt context and its downstream execution effects. UniMASO integrates heterogeneous assignment into iterative prompt optimization, making model choices depend on each candidate prompt, its topological position, downstream influence, and realized inference cost. This coupling matches model capacity to role requirements while optimizing the system-level quality–cost objective.

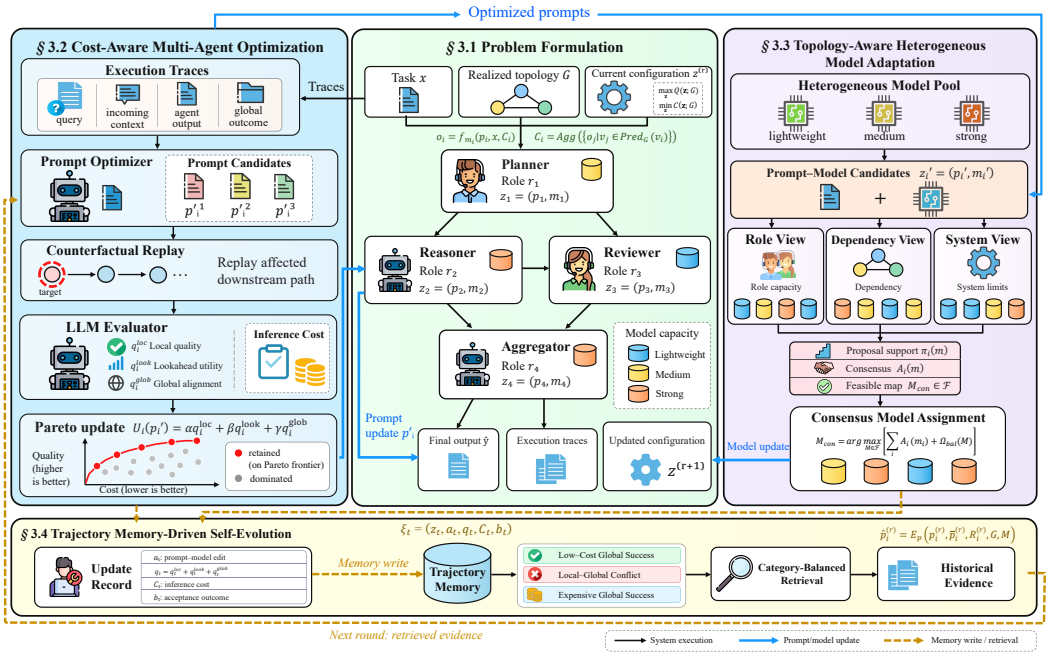

Figure 2: Overview of UniMASO. Cost-aware multi-agent optimization proposes and evaluates prompt candidates from execution traces; heterogeneous execution runs the current prompt–model configuration on the realized topology; topology-aware adaptation reconciles role, dependency, and system views to update model assignments; and trajectory memory records cross-level feedback and retrieves historical evidence for subsequent rounds.

3 Methodology

UniMASO alternates local search over agent configurations with topology-aware system coordination. As illustrated in Figure 2, local search proposes prompt–model candidates from interaction traces and evaluates them at multiple granularities, while system-level co-evolution reconciles these updates under the realized graph. Agents are visited in topological order, and each candidate is evaluated in context before accepted updates are stored in trajectory memory for later rounds. This ordering separates local exploration of prompt–model pairings from their global composition; the complete procedure is given in Algorithm 1 in Appendix B.

3.1 Problem Formulation

We model an N-agent system as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. An edge $(v_j, v_i) \in \mathcal{E}$ makes the output of v_j available to v_i as context. Each agent v_i has a prompt space $\mathcal{P}_i$ and is assigned a role prompt $p_i \in \mathcal{P}_i$ and an execution model m_i from the heterogeneous pool $\mathcal{M} = \{m^{(1)}, \ldots, m^{(K)}\}$. We write this assignment as $z_i = (p_i, m_i)$ and the full system configuration as $\mathbf{z} = (z_1, \ldots, z_N)$. Given input x, agent v_i produces

$$o_i = f_{m_i}(p_i, x, \mathcal{C}_i), \qquad \mathcal{C}_i = \mathrm{Agg}(\{o_j \mid v_j \in \mathrm{Pred}_{\mathcal{G}}(v_i)\}), \tag{1}$$

where $\mathrm{Agg}(\cdot)$ is protocol-specific aggregation. The full execution induces the system prediction $\hat{y} = \Phi(x; \mathcal{G}, \mathbf{z})$.

Given a validation set $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^{S}$ and task metric s, let $c(m_i, x_j; \mathbf{z}, \mathcal{G})$ denote the input–output token cost incurred by agent v_i on example x_j. We measure average task quality and average system cost as

$$Q(\mathbf{z}; \mathcal{G}) = \frac{1}{S} \sum_{j=1}^{S} s(\Phi(x_j; \mathcal{G}, \mathbf{z}), y_j), \qquad C(\mathbf{z}; \mathcal{G}) = \frac{1}{S} \sum_{j=1}^{S} \sum_{i=1}^{N} c(m_i, x_j; \mathbf{z}, \mathcal{G}). \tag{2}$$

The joint objective is to maximize task quality and minimize average token cost, i.e., $\max_{\mathbf{z}} Q(\mathbf{z}; \mathcal{G})$ and $\min_{\mathbf{z}} C(\mathbf{z}; \mathcal{G})$. Because local prompt or model changes propagate through $\mathcal{G}$, they can joint-

ly alter downstream context, task quality, and cost. UniMASO therefore evaluates prompt–model changes at the system level.

3.2 Cost-Aware Multi-Agent Optimization

Trace-guided candidate generation. An agent's prompt controls how it processes upstream information and what it passes to successors. For focal agent v_i, the optimizer $\mathcal{M}_{\text{opt}}$ samples traces containing inputs, contexts, outputs, and feedback, then proposes candidates from the current configuration. We first hold the model assignment fixed to measure prompt effects in context. Static topologies re-execute the focal agent and affected descendants, whereas routing changes trigger a full interaction; model alternatives are evaluated by the same procedure in Section 3.3.

Multi-granularity interaction evaluation. Local role fulfillment need not improve the final task. Following MASPO (Wang et al., 2026), we compare candidate and reference executions at three granularities: local validity, lookahead utility, and global alignment:

$$U_i(p') = \alpha q_i^{\text{loc}}(p') + \beta q_i^{\text{look}}(p') + \gamma q_i^{\text{glob}}(p'), \qquad \alpha + \beta + \gamma = 1. \tag{3}$$

Here, q_i^{loc} measures role fulfillment, q_i^{look} measures the effect on successor outputs, and q_i^{glob} measures the final response. Each is a normalized preference over the evaluation batch from $\mathcal{M}_{\text{eval}}$. Their weighted sum U_i guides candidate selection; it is distinct from the final task quality Q defined in Section 3.1.

Cost-aware beam search. Prompt changes can alter both token use and downstream cost. Building on MASPO's evolutionary beam search (Wang et al., 2026), let $C_i(p')$ be the mean system cost under candidate p' and $\eta_i(p') = C_i(p')/C_i(p_i)$ its relative cost. We expand a candidate when $U_i(p')\eta_i(p')^{-\lambda} > u_{\text{ref}}$, where $\lambda \geq 0$ is the cost weight and $u_{\text{ref}} = 1/2$. The threshold filters candidates whose cost-adjusted utility is below half of the reference. Eligible candidates and the current prompt form $\mathcal{P}_i^+$, from which we retain quality–cost nondominated candidates:

$$\mathcal{B}_i = \text{Select}_B\left(\text{ND}_{U_i\uparrow,\, C_i\downarrow}\left(\mathcal{P}_i^+\right)\right). \tag{4}$$

The operator ND removes candidates dominated in utility and cost, while Select_B keeps at most B candidates. Cost calibration controls expansion and nondominated selection preserves distinct quality–cost trade-offs. Retained candidates are re-evaluated after peer updates, and the best local state is passed to system coordination.

3.3 Topology-Aware Heterogeneous Model Adaptation

A fixed assignment limits the capabilities available at each node. We therefore extend the local candidates to prompt–model pairs $z_i' = (p_i', m_i')$, $m_i' \in \mathcal{M}$. Model capability and cost condition candidate generation, and every pair is executed and scored with the same multi-granularity utility and cost rule. The best local update is $\bar{z}_i = (\bar{p}_i, \bar{m}_i)$, which is then passed to system coordination.

Topology-conditioned multi-view proposals. Local updates are evaluated with the remaining configuration fixed, so their combination may change inter-agent behavior. System-level evolution resolves this mismatch: each optimizable logical agent forms a focal view conditioned on roles, dependencies, execution order, local search results, and model capability–cost information. It proposes a prompt $\hat{p}_j$ and complete assignment $\hat{\mathbf{M}}^{(j)}$; shared logical roles share variables, while deterministic routing and aggregation components remain fixed.

Proposal-entropy consensus. Focal views may disagree on a node's model. We measure their empirical support and weight it by proposal concentration:

$$\pi_i(m) = \frac{1}{J}\sum_{j=1}^{J} \mathbb{I}[\hat{m}_i^{(j)} = m], \qquad A_i(m) = \pi_i(m)(1 - \bar{H}_i). \tag{5}$$

Here, J is the number of focal views and $\bar{H}_i$ is the Shannon entropy of π_i normalized by $\log|\mathcal{M}_i|$. We set $\bar{H}_i = 0$ when only one model is feasible. Concentrated proposals therefore receive greater

weight in the configuration-level decision, which is solved subject to feasibility and capability–cost balance:

$$\mathbf{M}_{\text{con}} = \arg\max_{\mathbf{M}\in\mathcal{F}} \left[\sum_{i=1}^{N} A_i(m_i) + \Omega_{\text{bal}}(\mathbf{M}) \right]. \tag{6}$$

The feasible set $\mathcal{F}$ encodes model availability, role sharing, and routing constraints. Ω_{bal} regularizes concentration on one model or cost tier. The coordinated prompts are evaluated with the consensus assignment, and the resulting state anchors the next local-search round.

3.4 Trajectory Memory-Driven Self-Evolution

Local search and system coordination produce trajectories linking configuration changes to local, downstream, and global outcomes. We store these records as retrievable experience for later evolution.

Cross-level trajectory encoding. For optimization transition t, we record the preceding configuration, the proposed modification, and its evaluation:

$$\xi_t = (\mathbf{z}_t, a_t, \mathbf{q}_t, C_t, b_t), \qquad \mathbf{q}_t = (q_t^{\text{loc}}, q_t^{\text{look}}, q_t^{\text{glob}}). \tag{7}$$

Here, $\mathbf{z}_t$ is the preceding configuration, a_t the modification, $\mathbf{q}_t$ the multi-granularity feedback, C_t the execution cost, and b_t the acceptance outcome. We group records into low-cost global successes, local successes with global failures, and high-cost global successes, providing examples of efficient changes, coordination conflicts, and effective but costly alternatives.

Category-balanced retrieval. To prevent frequent categories from dominating nearest-neighbor retrieval, we retrieve within each category. The normalized embedding of the current prompt selects up to K_{mem} trajectories per category by similarity. Each record contains the configuration, modification, and evaluation feedback.

Experience-guided evolution. At round r, retrieved experience is combined with the current and locally optimized prompts:

$$\hat{p}_i^{(r)} = \mathcal{E}_p\left(p_i^{(r)}, \bar{p}_i^{(r)}, \mathcal{R}_i^{(r)}, \mathcal{G}, \mathcal{M}\right). \tag{8}$$

The evolution operator $\mathcal{E}_p$ interprets these records under current dependencies and available models. Low-cost successes suggest reusable changes, local–global conflicts expose downstream effects, and high-cost successes provide targets for cost reduction. The updated configuration is executed and its trajectory is written back to memory.

4 Experiments

4.1 Experimental Setup

Benchmarks, Models, and Topologies. We evaluate UniMASO on AGIEval-MATH at difficulty level 5 (Zhong et al., 2024), GPQA-Diamond (Rein et al., 2024), BFCL-V3 (Yan, 2025), LiveCodeBench v6 (Jain et al., 2025), and HotpotQA (Yang et al., 2018). LiveCodeBench uses problems released between February 1 and May 1, 2025. We test Independent, Centralized, Sequential, Decentralized, and Dynamic topologies to vary information flow and downstream dependencies. The model pool comprises Qwen3.5-4B, Qwen3.5-9B, Qwen3.5-27B (Qwen Team, 2026a), DeepSeek-V4-Flash, and DeepSeek-V4-Pro (DeepSeek-AI, 2026). Dataset details and topology configurations appear in Appendices A and D, respectively.

Baselines. TPE (Tree-structured Parzen Estimator), used in MIPRO (Opsahl-Ong et al., 2024) and MASS (Zhou et al., 2026), and SPO (Self-Supervised Prompt Optimization) (Xiang et al., 2025) are prompt optimizers. We compare UniMASO with DeepSeek-V4-Pro single-agent baselines (direct, CoT, +TPE, +SPO) and MAS baselines (unoptimized Centralized MAS, TPE, SPO, MASPOB (Hong et al., 2026), and MASPO (Wang et al., 2026)). MAS prompt-only baselines optimize role prompts while keeping topology and model assignments fixed. Across topologies, five homogeneous configurations are compared with UniMASO's heterogeneous assignment under identical evaluation and deployment-cost accounting.

Table 2: Task quality and deployment cost under the centralized topology. Prompt-only baselines use homogeneous models; UniMASO jointly optimizes prompts and model assignments.

Model	Method	AGIEval		GPQA		BFCL		LiveCodeBench		HotpotQA		Avg.	
		Acc.	Cost	Acc.	Cost	Acc.	Cost	pass@1	Cost	EM	Cost	Result	Cost
Qwen3.5-4B	Vanilla MAS	62.11	7.3	67.17	5.3	88.40	4.8	18.32	3.8	58.00	5.8	58.80	5.38
	+TPE	59.77	8.2	66.67	6.1	88.80	5.4	19.08	4.2	58.20	6.5	58.50	6.08
	+SPO	63.28	8.3	69.70	5.9	88.60	5.5	20.61	4.3	58.00	6.6	60.03	6.12
	+MASPOB	63.67	8.2	68.69	5.8	89.40	5.3	19.85	4.3	58.60	6.5	60.04	6.02
	+MASPO	63.28	8.2	69.19	5.9	89.60	5.5	21.37	4.4	58.40	6.4	60.36	6.08
Qwen3.5-9B	Vanilla MAS	71.09	14.3	71.72	12.2	90.40	9.1	22.14	8.1	56.80	10.5	62.43	10.84
	+TPE	71.88	15.8	72.22	13.6	87.80	10.2	21.37	9.1	57.20	12.0	62.09	12.14
	+SPO	71.48	16.3	73.23	13.7	90.80	10.2	25.19	8.9	57.00	11.6	63.54	12.14
	+MASPOB	72.66	16.4	72.73	13.7	91.00	10.0	24.43	9.3	57.60	11.6	63.68	12.20
	+MASPO	72.27	16.2	73.74	13.6	91.40	10.1	25.19	9.0	57.40	11.8	64.00	12.18
Qwen3.5-27B	Vanilla MAS	75.78	56.2	77.27	41.0	90.40	38.7	29.77	26.4	60.80	48.5	66.80	42.18
	+TPE	76.17	64.2	78.28	46.2	91.00	44.5	30.53	29.3	63.80	54.1	67.96	47.66
	+SPO	73.05	64.1	77.78	46.1	90.60	43.6	29.01	30.0	61.20	54.4	66.33	47.64
	+MASPOB	76.56	62.1	79.29	46.9	91.40	43.6	32.06	29.7	61.40	55.6	68.14	47.58
	+MASPO	77.34	63.5	78.79	45.7	91.60	43.4	31.30	30.1	61.80	54.5	68.17	47.44
DeepSeek-V4-Flash	Vanilla MAS	72.27	57.3	70.20	24.8	92.20	73.1	38.17	39.6	59.20	100.2	66.41	59.00
	+TPE	72.66	64.2	67.68	27.4	92.60	81.5	38.93	44.6	59.60	112.7	66.30	66.08
	+SPO	73.05	64.1	70.71	27.7	92.40	81.5	40.46	43.8	62.00	114.7	67.72	66.36
	+MASPOB	73.83	64.0	71.72	28.0	92.80	82.8	39.69	45.5	60.00	114.5	67.61	66.96
	+MASPO	73.44	63.3	72.22	28.1	93.20	83.4	41.22	44.0	59.80	113.1	67.97	66.38
DeepSeek-V4-Pro	Vanilla MAS	76.17	217.3	78.28	103.4	92.00	195.1	32.82	112.0	61.60	295.0	68.17	184.55
	+TPE	76.95	249.3	78.79	118.6	92.20	219.9	30.53	128.7	62.00	330.7	68.09	209.44
	+SPO	76.56	239.2	79.80	116.6	94.80	219.6	33.59	125.2	61.80	335.8	69.31	207.28
	+MASPOB	77.34	246.9	79.29	117.2	92.80	219.2	35.88	125.8	62.60	338.4	69.58	209.50
	+MASPO	78.12	249.6	80.30	115.2	92.60	216.6	35.11	124.0	62.80	336.0	69.79	208.28
DeepSeek-V4-Pro	Single Agent	74.61	36.5	76.77	23.0	92.80	25.9	36.64	16.9	52.60	55.2	66.68	31.50
	+CoT	75.39	42.3	75.76	26.5	85.00	26.6	38.93	21.9	61.00	69.8	67.22	37.41
	+TPE	71.09	37.2	73.74	16.1	92.60	26.5	36.64	16.3	61.20	54.2	67.05	30.04
	+SPO	75.39	39.9	73.74	16.9	93.00	26.4	38.93	16.0	60.20	54.1	68.25	30.65
Mixed Models	UniMASO	82.42	56.9	78.28	33.0	92.40	12.2	35.11	13.3	63.80	39.3	70.40	30.92

Implementation Details. Unless otherwise stated, UniMASO runs three optimization rounds with per-round search depths of 3–2–2. Task agents use temperature 0.0, and optimizer and evaluator settings are fixed across methods. Following MASPO (Wang et al., 2026), we maintain an optimization pool of $|\mathcal{D}_{\text{opt}}| = 50$ per benchmark and randomly sample 10 instances per iteration. Memory retrieval selects up to $K_{\text{mem}} = 2$ trajectories per category, and the cost weight is set to $\lambda = 0.2$. The selected configuration is evaluated on each benchmark's full evaluation file. Further implementation details and prompt templates appear in Appendices C and F, respectively.

Metrics. We report accuracy for AGIEval-MATH, GPQA-Diamond, and BFCL-V3, Pass@1 for LiveCodeBench, and exact match for HotpotQA. The overall score is their unweighted mean. We define task-agent deployment cost as $\text{Cost}(\mathbf{z}) = \sum_i (c^{\text{in}}_{m_i} T^{\text{in}}_i + c^{\text{out}}_{m_i} T^{\text{out}}_i)$, where T^{in}_i and T^{out}_i are the input and output token counts of agent i, and $c^{\text{in}}_{m_i}$ and $c^{\text{out}}_{m_i}$ are fixed relative model-cost coefficients derived from the Qwen AI model pricing reference. We report the cost of executing the selected configuration in millions of weighted-token units (M). Further details appear in Appendix E.

4.2 Main Results

UniMASO achieves the strongest centralized quality–cost trade-off. Table 2 compares UniMASO with prompt-only baselines on the same centralized topology. These baselines optimize role prompts while keeping model assignments homogeneous. Within each model group, the best prompt-only variant improves on its unoptimized counterpart but incurs higher deployment cost. DeepSeek-V4-Pro + MASPO is the strongest prompt-only configuration, scoring 69.79 at a cost of 208.28M. UniMASO raises the average score to 70.40 and lowers cost to 30.92M, a gain of 0.61 points with an 85.2% reduction in token-weighted deployment cost. It achieves this operating point by jointly adapting prompts and model assignments to match model capacity to agent roles. UniMASO leads on AGIEval-MATH (82.42 vs. 78.12) and HotpotQA (63.80 vs. 62.80), while homogeneous configurations score higher on GPQA-Diamond, BFCL-V3, and LiveCodeBench.

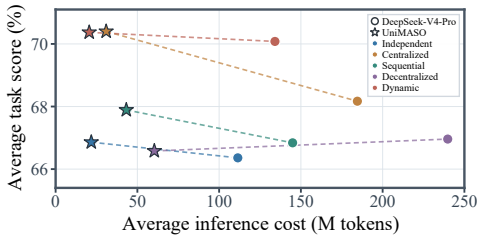

Figure 3: Quality–cost trade-off by topology.

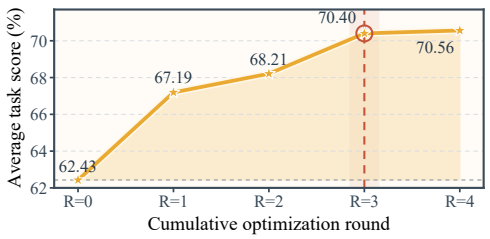

Figure 4: Mean score by optimization round.

UniMASO's quality–cost advantage transfers across execution topologies. Figure 3 plots the paired quality–cost points for DeepSeek-V4-Pro and UniMASO across Independent, Centralized, Sequential, Decentralized, and Dynamic execution. UniMASO moves the comparison toward lower deployment cost in all five topologies and improves the mean score in four. Decentralized execution is the only quality decrease, trading 66.96 for 66.58 while reducing cost from 239.80M to 60.37M. The Dynamic topology gives a representative example of the same pattern: UniMASO reaches 70.37 at 20.56M, compared with 70.08 at 134.13M for the homogeneous reference. Full per-task scores and costs appear in Table 7 (Appendix D).

5 Analysis

We analyze the contributions of UniMASO's components, the sensitivity of its optimization settings, and the behavior of the resulting MAS. Unless stated otherwise, comparisons use centralized execution and the protocol in Section 4.1. Table 3 reports component ablations and parameter sensitivity.

5.1 Ablation Study of UniMASO

Effect of Joint Prompt–Model Optimization. We test whether role prompts and model assignments should be optimized together rather than in consecutive stages. The control first optimizes prompts and then assigns models, whereas UniMASO evaluates prompt–model pairs during joint search under the same centralized topology (Table 3). The staged control lowers the average score from 70.40 to 66.39 and raises deployment cost from 30.92M to 64.15M. The loss spans all five benchmarks; on GPQA-Diamond, accuracy falls from 78.28 to 71.72 while cost rises from 33.02M to 70.24M. A prompt chosen before model reassignment may fit the original executor better than the model ultimately assigned to that role. Jointly evaluating both decisions is therefore consistent with the observed improvement in quality and cost.

Effect of Topology-Aware Co-Evolution. We examine whether locally selected updates remain effective after interacting agents adopt their new prompts and models. The ablation removes system-level co-evolution while retaining local candidate search on the centralized graph (Table 3). Without cross-agent coordination, the average score falls from 70.40 to 68.49, whereas deployment cost changes only from 30.92M to 31.20M. GPQA-Diamond shows the largest score drop (4.54 points), followed by AGIEval-MATH (2.34 points) and LiveCodeBench (2.29 points); HotpotQA improves slightly (0.20 points). The near-constant cost suggests that the main difference is how updates interact across dependent roles rather than a systematic shift toward more expensive execution. Coordinating local proposals is useful when a change at one agent affects its successors.

Effect of Cost-Aware Pareto Search. We assess whether candidate selection needs to account for execution cost as well as task utility. UniMASO uses relative cost when expanding candidates and retains nondominated quality–cost states; the ablation removes this cost-aware Pareto search (Table 3). The average score decreases from 70.40 to 69.42, while deployment cost increases from 30.92M to 62.40M. On BFCL-V3, the ablation raises accuracy slightly from 92.40 to 92.60 but almost doubles cost from 12.21M to 24.53M. AGIEval-MATH shows a similar cost increase, from 56.86M to 110.81M, without an accuracy gain. These comparisons show why task quality alone does not identify an efficient configuration. Cost-aware expansion and retention improve the average operating point across benchmarks.

Table 3: UniMASO ablations and parameter sensitivity under centralized execution.

Method	AGIEval Acc.	AGIEval Cost	GPQA Acc.	GPQA Cost	BFCL Acc.	BFCL Cost	LiveCodeBench pass@1	LiveCodeBench Cost	HotpotQA EM	HotpotQA Cost	Avg. Result	Avg. Cost
Reference: Proposed Framework												
UniMASO (Full)	82.42	56.86	78.28	33.02	92.40	12.21	35.11	13.27	63.80	39.26	70.40	30.92
(I) Component Ablations												
w/o Memory	80.08	51.35	76.77	31.58	91.60	13.18	29.01	13.35	61.80	23.04	67.85	26.50
w/o Co-Evolution	80.08	53.31	73.74	37.57	91.80	13.12	32.82	13.04	64.00	38.96	68.49	31.20
w/o Joint Optimization	78.52	121.38	71.72	70.24	89.20	18.07	30.53	24.89	62.00	86.17	66.39	64.15
w/o Cost-Aware Pareto Search	82.03	110.81	76.26	73.28	92.60	24.53	33.59	26.22	62.60	77.16	69.42	62.40
(II) Optimization Depth per Round (R1–R2–R3; default: 3–2–2)												
Depth = 3–3–3	82.03	109.49	77.78	79.97	92.20	34.05	35.11	24.56	63.60	92.93	70.14	68.20
Depth = 3–2–1	78.12	52.37	74.24	31.94	91.40	13.07	28.24	9.94	62.40	20.68	66.88	25.60
(III) Sensitivity to Retrieved Memory Size (K_{mem}, default: 2)												
$K_{\mathrm{mem}} = 1$	80.86	44.65	77.78	22.92	90.60	9.99	29.01	12.26	64.00	50.68	68.45	28.10
$K_{\mathrm{mem}} = 3$	80.86	90.15	77.78	69.55	92.00	24.96	35.88	22.51	62.80	76.83	69.86	56.80
$K_{\mathrm{mem}} = 4$	79.30	138.32	75.76	69.87	91.00	18.20	28.24	28.07	61.20	67.04	67.10	64.30
(IV) Sensitivity to Cost Weight (λ, default: 0.2)												
$\lambda = 0$	81.64	110.24	77.78	73.18	92.00	18.65	34.35	21.47	63.60	80.36	69.87	60.78
$\lambda = 0.1$	81.25	69.82	77.78	43.26	91.80	15.72	35.88	20.64	63.40	76.13	70.02	45.11
$\lambda = 0.5$	82.03	81.47	77.27	49.38	92.20	12.16	34.35	23.94	63.60	79.12	69.89	49.21
$\lambda = 1.0$	80.86	74.25	77.78	45.17	92.00	19.18	35.11	21.73	63.40	73.84	69.83	46.83

Effect of Trajectory Memory. We test whether stored trajectories improve optimization and transfer across related tasks. Removing memory lowers the average score from 70.40 to 67.85 and cost from 30.92M to 26.50M (Table 3); LiveCodeBench drops most, from 35.11 to 29.01. Table 4 compares no memory, own-task records, and cross-task records on AGIEval-MATH and GPQA-Diamond. Cross-task records raise scores from 80.08 to 81.64 and from 76.77 to 77.78, respectively, retaining roughly two-thirds of the gains from own-task memory. Cross-task costs of 52.93M and 29.43M are below own-task costs of 56.86M and 33.02M. Memory transfers across tasks, while own-task records remain most accurate.

Table 4: Cross-task memory transfer.

Target	Memory	Acc.	Cost
AGIEval-MATH	None	80.08	51.35
	Own	82.42	56.86
	Other	81.64	52.93
GPQA-Diamond	None	76.77	31.58
	Own	78.28	33.02
	Other	77.78	29.43

5.2 Optimization Dynamics and Quality–Cost Trade-offs

Effect of Optimization Rounds. We vary the number of complete optimization rounds to determine when additional adaptation yields diminishing returns. Figure 4 tracks the average task score through four rounds; the selected configuration uses three. The score reaches 67.19 after the first round, 68.21 after the second, and 70.40 after the third. A fourth round adds only 0.16 points, reaching 70.56. Thus, the first three rounds capture 98.0% of the four-round gain, making round three a practical stopping point for these tasks.

Effect of Per-Round Search Budget. We separately vary the number of candidate-search iterations allocated to each of the three rounds. The default 3–2–2 schedule is compared with 3–3–3 and 3–2–1 while trajectory memory remains enabled (Table 3). With 3–3–3, the selected configuration scores 70.14 at 68.20M, versus 70.40 at 30.92M for the default: deployment cost more than doubles without a score gain. The 3–2–1 schedule lowers cost to 25.60M but also lowers the score to 66.88. Since the number of rounds is fixed, these differences reflect the per-round budget. Among these schedules, 3–2–2 yields the best quality–cost balance.

Effect of Retrieved Memory Size. We vary how many trajectories are retrieved from each memory category to examine the amount of history used during evolution. The default is $K_{\mathrm{mem}} = 2$; we also evaluate one, three, and four records per category under centralized execution (Table 3). With one record, the average score falls from 70.40 to 68.45, while cost declines from 30.92M to 28.10M. Three records yield 69.86 at 56.80M, and four yield 67.10 at 64.30M. Although three records improve LiveCodeBench from 35.11 to 35.88, neither larger setting improves the overall score. Two records give the strongest average without the higher deployment costs of larger retrieval contexts.

Effect of Cost Weight. We vary the search weight λ to determine how strongly relative execution cost should affect candidate expansion. The default $\lambda = 0.2$ is compared with 0, 0.1, 0.5, and 1.0 in Table 3. Removing the cost term yields 69.87 at 60.78M, versus 70.40 at 30.92M for the default. Weights 0.1, 0.5, and 1.0 yield 70.02 at 45.11M, 69.89 at 49.21M, and 69.83 at 46.83M, respectively. The default thus attains both the highest score and the lowest deployment cost in this sweep. The nonmonotonic response reflects which prompt–model candidates survive under each weight.

5.3 Coordination Efficiency and Model Behavior

Multi-Agent Coordination and Efficiency. We assess UniMASO's coordination and cost per correct answer. Communication and Planning Scores are reported on a percentage scale using MultiAgentBench (Zhu et al., 2025). Cost-of-Pass takes the unweighted mean of deployment cost per correct answer across five tasks, in millions of weighted-token units (Erol et al., 2026) (Table 5). Compared with centralized +MASPO, UniMASO raises Communication Score from 72.84 to 73.76 and reduces Cost-of-Pass from 1.241 to 0.184 (85.2%), while maintaining a comparable Planning Score (67.68 versus 68.41). UniMASO thus improves agent communication and substantially lowers the cost per correct answer.

Table 5: Coordination and efficiency.

Method	Coordination		Efficiency
	Comm. ↑	Plan. ↑	Cost-of-Pass ↓
Centralized MAS	68.45	64.39	1.154
+TPE	71.81	66.86	1.357
+SPO	69.90	64.15	1.271
+MASPOB	71.54	67.08	1.245
+MASPO	72.84	**68.41**	1.241
UniMASO	**73.76**	67.68	**0.184**

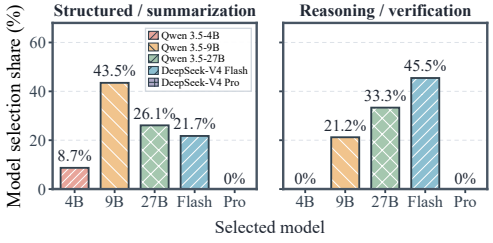

Figure 5: Model selection by agent role.

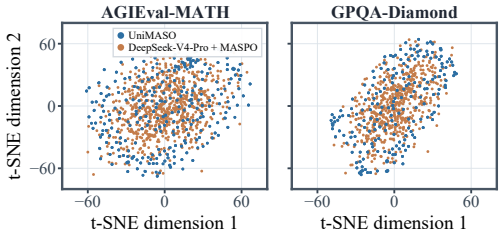

Figure 6: Primary-agent response embeddings.

Role-Conditioned Model Allocation. We examine whether UniMASO matches model capability to the demands of different agent roles. We select two groups of agents for comparison: those responsible for structured processing and summarization and those responsible for reasoning and verification (Figure 5). Qwen3.5-9B is selected most often in the first group (43.5%), whereas DeepSeek-V4-Flash dominates the second (45.5%). The combined share of Qwen3.5-27B and DeepSeek-V4-Flash rises from 47.8% to 78.8% between the groups. This shift shows that UniMASO allocates greater model capacity to roles with higher reasoning demands.

Generation Diversity. We examine how joint prompt–model adaptation affects the diversity of primary-agent responses. On AGIEval-MATH and GPQA-Diamond, we embed outputs from agents 0–2 of UniMASO and DeepSeek-V4-Pro + MASPO using Qwen3.7-Text-Embedding (Qwen Team, 2026b) and visualize them with t-SNE (Figure 6). UniMASO's responses span a broader region in both projections, particularly on GPQA-Diamond. This broader coverage provides qualitative evidence that UniMASO increases the diversity of primary-agent responses.

6 Conclusion

We introduced UniMASO to jointly adapt prompts and model assignments in heterogeneous MAS. It combines cost-aware search over local, downstream, and global feedback with topology-aware coevolution and cross-level trajectory memory. Across five benchmarks and five execution topologies, UniMASO achieves favorable quality–cost trade-offs relative to homogeneous configurations. Ablations support the contributions of joint adaptation, coordination, cost-aware selection, and memory; cross-task results show that trajectory experience transfers between related reasoning tasks. These findings favor joint adaptation for efficient multi-agent collaboration.

AI USE STATEMENT

We used generative AI tools to modify scientific figures and to edit the manuscript to improve its clarity and readability. We did not use generative AI tools to generate synthetic data or develop mathematical proofs, as these tasks were not applicable to this work. We reviewed all AI-assisted content and take responsibility for the final text, figures, claims, and artifacts in this work.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0e9e708b6f48e14fd0ac29e167413f76-Abstract-Conference.html.

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

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Matei Zaharia, James Zou, and Ion Stoica. Optimizing model selection for compound ai systems. *arXiv preprint arXiv:2502.14815*, 2025. URL https://arxiv.org/abs/2502.14815.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/578e65cdee35d00c708d4c64bce32971-Abstract-Conference.html.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms. In *Advances in Neural Information Processing Systems*, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/83ba7056bce2c3c3c27e17397cf3e1f0-Abstract-Conference.html.

DeepSeek-AI. DeepSeek-V4 Model Card, 2026. URL https://fe-static.deepseek.com/chat/transparency/deepseek-V4-model-card-EN.pdf.

Mehmet Hamza Erol, Batu El, Mirac Suzgun, Mert Yuksekgonul, and James Y. Zou. Cost-of-pass: An economic framework for evaluating language models. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/399d54cce01e782192b3d7ecbe607a96-Abstract-Conference.html.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rockt"aschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 13481–13544. PMLR, 2024. URL https://proceedings.mlr.press/v235/fernando24a.html.

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

Zhi Hong, Qian Zhang, Jiahang Sun, Zhiwei Shang, Mingze Kong, Xiangyi Wang, Yao Shu, and Zhongxiang Dai. MASPOB: Bandit-based prompt optimization for multi-agent systems with graph neural networks. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026. URL https://arxiv.org/abs/2603.02630.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/94074dd5a072d28ff75a76dabed43767-Abstract-Conference.html.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large scale language model society. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper/2023/hash/a3621ee907def47c1b952ade25c67698-Abstract-Conference.html.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8sSqNntaMr.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.525. URL https://aclanthology.org/2024.emnlp-main.525/.

Qwen Team. Qwen3.5: Towards Native Multimodal Agents, 2026a. URL https://qwen.ai/blog?id=qwen3.5. Official model cards are available on Hugging Face.

Qwen Team. Qwen3.7-Text-Embedding, 2026b. URL https://www.qianwenai.com/models/qwen3.7-text-embedding. Official model page.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Zhexuan Wang, Xuebo Liu, Li Wang, Zifei Shan, Yutong Wang, Zhenxi Song, and Min Zhang. MASPO: Joint prompt optimization for LLM-based multi-agent systems. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2308.08155.

Jinyu Xiang, Jiayi Zhang, Zhaoyang Yu, Xinbing Liang, Fengwei Teng, Jinhao Tu, Fashen Ren, Xiangru Tang, Sirui Hong, Chenglin Wu, and Yuyu Luo. Self-supervised prompt optimization. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 9017–9041. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.findings-emnlp.479/.

Fanjia Yan. A function calling perspective on scalable large language model agent evaluation. Master's thesis, EECS Department, University of California, Berkeley, 2025. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-184.html.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380. Association for Computational Linguistics, 2018. URL https://aclanthology.org/D18-1259/.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.757. URL https://aclanthology.org/2025.acl-long.757/.

Zheyuan Zhang, Lin Ge, Hongjiang Li, Weicheng Zhu, Chuxu Zhang, and Yanfang Ye. Mapro: Recasting multi-agent prompt optimization as maximum a posteriori inference. In *Findings of the Association for Computational Linguistics: EACL 2026*. Association for Computational Linguistics, 2026a. URL https://aclanthology.org/2026.findings-eacl.233/.

Zhiheng Zhang, Yuanzhe Zhang, Bohan Yu, Daojian Zeng, Kang Liu, and Jun Zhao. Heterodesigner: Automated design of multi-agent systems with heterogeneous LLMs. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27553–27574. Association for Computational Linguistics, 2026b. doi: 10.18653/v1/2026.acl-long.1272. URL https://aclanthology.org/2026.acl-long.1272/.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. AGIEval: A human-centric benchmark for evaluating foundation models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 2299–2314, Mexico City, Mexico, June 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-naacl.149/.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulič, Anna Korhonen, and Sercan O. Arik. Multi-agent design: Optimizing agents with better prompts and topologies. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/1ab4e0e8f35078109bc78d7b465d306f-Abstract-Conference.html.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8580–8622. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.421. URL https://aclanthology.org/2025.acl-long.421/.

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

A Benchmark and Evaluation Protocol

A.1 Task Datasets

We select five benchmarks to cover mathematical reasoning, graduate-level scientific question answering, executable function calling, competitive programming, and multi-hop question answering. We retain each benchmark's task format and use the evaluation metric specified in Section 4.1. The dataset-specific roles in our experiments are described below.

AGIEval-MATH. AGIEval is a bilingual, human-centric benchmark assembled from official and public admission and qualification examinations, including college entrance tests, law-school admissions, lawyer qualification tests, civil-service exams, and mathematics competitions (Zhong et al., 2024). Its MATH task contains English-language competition problems drawn from the AMC and AIME examinations. For the experiments reported in this paper, we use the level-5 AGIEval-MATH subset distributed in our evaluation manifest, which contains 256 instances. A prediction is counted as correct when its final extracted answer matches the reference under our accuracy protocol.

Table 6: Final-evaluation sample counts and metrics by benchmark.

Dataset	Samples	Metric
AGIEval-MATH	256	Accuracy
GPQA-Diamond	198	Accuracy
BFCL-V3	500	Accuracy
LiveCodeBench-v6	131	Pass@1
HotpotQA	500	Exact match
Total	**1,585**	–

GPQA-Diamond. GPQA is a graduate-level multiple-choice benchmark of 448 questions written by domain experts in biology, physics, and chemistry (Rein et al., 2024). The Diamond subset contains 198 questions selected for high answer quality and difficulty, with questions designed to resist solution by straightforward web search. We use GPQA-Diamond to test scientific reasoning under a fixed four-option format. The final option selected by the multi-agent system is compared with the gold option, and the reported score is instance-level accuracy.

BFCL-V3. The Berkeley Function-Calling Leaderboard (BFCL) evaluates whether a model can select tools and construct valid function calls from natural-language requests and function specifications (Yan, 2025). Version 3 retains the single-turn benchmark while adding multi-turn and multi-step interactions in which tool calls modify and query an executable state. Its evaluation combines structured call checking with execution-based verification, and the multi-turn setting additionally checks both the resulting system state and the required execution path. We use BFCL-V3 to measure structured tool use and report the benchmark accuracy returned by its evaluator.

LiveCodeBench. LiveCodeBench is a continuously updated coding benchmark that collects recent problems from LeetCode, AtCoder, and Codeforces, reducing the risk that test items were seen during model training (Jain et al., 2025). In addition to code generation, the benchmark supports code execution, test-output prediction, and self-repair scenarios. We use the v6 problems in the date window specified in Section 4.1 and evaluate generated programs with executable tests. The reported metric is pass@1, so an instance is correct only when the single submitted program passes the benchmark checker.

HotpotQA. HotpotQA contains approximately 113,000 Wikipedia-based question–answer pairs designed for diverse and explainable multi-hop reasoning (Yang et al., 2018). Questions require combining evidence from multiple supporting documents, and the dataset provides sentence-level supporting facts as well as comparison questions between entities. We use HotpotQA to test evidence aggregation and cross-agent context integration. Following the protocol in Section 4.1, we report normalized exact match between the system's final answer and the reference answer.

Table 6 reports the final-evaluation sample count and metric for each benchmark. The five benchmarks comprise 1,585 evaluation instances per topology, yielding 7,925 instance–topology evaluations across all five topologies.

A.2 Optimization Baselines

We compare UniMASO with prompt-optimization methods while keeping the execution topology, task-agent model assignment, evaluation metrics, and deployment-cost accounting fixed within each comparison. The baseline methods are described below according to their original optimization principles; the controlled comparison in Table 2 uses their prompt optimization variants under the centralized topology.

Unoptimized Centralized MAS. This baseline uses the same centralized execution graph as UniMASO but keeps the manually initialized role prompts unchanged. All task agents use one homogeneous model selected from the five-model pool. It isolates the benefit of optimization from the benefit of multi-agent execution itself.

TPE. We apply a Tree-structured Parzen Estimator (TPE), as used in MIPRO (Opsahl-Ong et al., 2024) and MASS (Zhou et al., 2026), to optimize role prompts in the centralized MAS. Candidate prompts are selected using end-to-end task feedback while the execution topology and agent model assignments remain fixed.

SPO. Self-Supervised Prompt Optimization (SPO) removes the need for external reference answers during prompt search. It compares the outputs produced by competing prompts with an LLM evaluator and uses an LLM optimizer to improve the prompt according to task requirements (Xiang et al., 2025). In our comparison, SPO is applied to the role prompts of the fixed centralized system. The task-agent model remains homogeneous, so any improvement is attributable to self-supervised prompt selection rather than model allocation.

MASPOB. MASPOB formulates multi-agent prompt optimization as a sample-efficient bandit problem. Its search combines upper-confidence-bound exploration, a graph neural network surrogate for topology-aware prompt representations, and coordinate ascent to reduce the combinatorial search space (Hong et al., 2026). We use MASPOB as a topology-aware prompt-only baseline: the centralized graph is fixed and the execution model associated with each role is not optimized.

MASPO. MASPO jointly optimizes prompts across interacting agents rather than scoring each prompt in isolation. Its evaluation considers the effect of a prompt on local behavior, downstream agents, and the final system output, and its search uses an evolutionary beam-search procedure to explore prompt configurations under a system-level objective (Wang et al., 2026). Our MASPO baseline keeps the same homogeneous model at all task-agent nodes. UniMASO extends this prompt-level setting by adding model assignment to the search state and by including relative inference cost in candidate selection.

B Optimization Algorithm

Algorithm 1 summarizes UniMASO. The method alternates cost-aware local beam search with topology-aware, memory-guided co-evolution. trajectories are recorded during evaluation and retrieved at round boundaries.

Algorithm 1 UniMASO: Joint Prompt–Model Optimization

Input: topology $\mathcal{G}$, optimization set $\mathcal{D}_{\text{opt}}$, model pool $\mathcal{M}$, initial configuration $\mathbf{Z}^{(0)}$, beam width B, rounds R, search budgets $\{T_r\}_{r=1}^{R}$, cost weight λ, and memory size K_{mem}.
Initialize: $\mathbf{Z} \leftarrow \mathbf{Z}^{(0)}$, $\mathcal{H} \leftarrow \varnothing$, and $\mathcal{B}_i \leftarrow \{(z_i^{(0)}, 0)\}$ for every optimizable agent v_i.
for $r = 1$ to R **do**
  for each optimizable agent v_i in TOPOLOGICALSORT($\mathcal{G}$) **do**
    if BEAMREFRESH is enabled and $\mathcal{B}_i$ is nonempty **then**
      Re-evaluate and re-rank existing beam nodes under the updated partner-agent configuration.
    end if
    for $t = 1$ to T_r **do**
      Sample traces $\mathcal{T}_{i,r,t}$ from $\mathcal{D}_{\text{opt}}$ and generate candidates

$$\mathcal{C}_{i,r,t} \leftarrow \mathcal{E}_{\text{loc}}(\mathcal{B}_i, \mathcal{T}_{i,r,t}, \mathcal{G}, \mathcal{M}).$$

      for each $z_i' = (p_i', m_i') \in \mathcal{C}_{i,r,t}$ **do**
        Evaluate the affected graph and obtain $(q_i^{\text{loc}}, q_i^{\text{look}}, q_i^{\text{glob}}, C_i)$.
        Compute

$$U_i(z_i') = \alpha q_i^{\text{loc}} + \beta q_i^{\text{look}} + \gamma q_i^{\text{glob}}, \qquad \eta_i(z_i') = \frac{C_i(z_i')}{C_i(z_i)}.$$

        If $U_i(z_i')\eta_i(z_i')^{-\lambda} > u_{\text{ref}}$, mark z_i' eligible and record its trajectory in $\mathcal{H}$.
      end for
      Update the beam:

$$\mathcal{B}_i \leftarrow \text{SELECT}_B\left(\text{ND}_{U_i\uparrow, C_i\downarrow}(\mathcal{B}_i \cup \mathcal{C}_{i,r,t})\right).$$

      Synchronize the best eligible local candidate with $\mathbf{Z}$.
    end for
    Store the best local state $\bar{z}_i^{(r)} \leftarrow \text{BEST}(\mathcal{B}_i)$.
  end for
  if COEVOLVER is enabled **then**
    for each optimizable logical agent v_i **do**
      Retrieve category-balanced trajectories

$$\mathcal{R}_i^{(r)} \leftarrow \text{RETRIEVE}_{\text{bal}}(\mathcal{H}, p_i^{(r)}, K_{\text{mem}})$$

      and generate $(\widehat{p}_i^{(r)}, \widehat{\mathbf{M}}^{(i,r)})$ using topology $\mathcal{G}$ and the current configuration.
    end for
    Merge model proposals:

$$\mathbf{M}_{\text{con}}^{(r)} \leftarrow \text{PROPOSALENTROPYCONSENSUS}(\{\widehat{\mathbf{M}}^{(i,r)}\}, \mathcal{G}, \mathcal{M}).$$

    Set $\mathbf{P}^{(r+1)} \leftarrow \{\widehat{p}_i^{(r)}\}$ and

$$\mathbf{M}^{(r+1)} \leftarrow \mathbf{M}_{\text{con}}^{(r)}.$$

  else
    Set $(\mathbf{P}^{(r+1)}, \mathbf{M}^{(r+1)})$ to the best local configuration.
  end if
  Update $\mathbf{Z} \leftarrow (\mathbf{P}^{(r+1)}, \mathbf{M}^{(r+1)})$ and reset each beam to its corresponding prompt–model state.
end for
Evaluate the final configuration on the evaluation file and **return** $\mathbf{Z}^{\star}$.

C Implementation Details

All optimization runs use a random seed of 42. For UniMASO, we initialize each optimizable agent's model assignment to Qwen3.5-9B. The optimizer and evaluator use Qwen3.5-27B, and the Co-Evolver uses DeepSeek-V4-Flash. Following MASPO (Wang et al., 2026), we set the per-agent beam width to $B = 2$ and the weights of the local, lookahead, and global utility terms to $(\alpha, \beta, \gamma) = (0.4, 0.4, 0.2)$, respectively.

D Multi-Agent Topologies and Full Results

We consider five execution topologies. Their definitions below distinguish logical roles from mechanical aggregation modules: only LLM-backed task roles are included in model assignment and deployment-time task-agent cost.

D.1 Topology Definitions

Independent topology. The Independent topology executes three task agents on the same input without passing intermediate messages between them. Each agent produces an independent candidate solution, after which a mechanical aggregation rule combines the three outputs. Because the final aggregator does not invoke an LLM, it is not an optimizable model-assignment variable. This topology therefore measures the value of independent model and reasoning diversity without learned coordination.

Centralized topology. The Centralized topology contains three specialist agents followed by a terminal summarizer. The specialists process the task in parallel, and their outputs are sent to the summarizer, which is responsible for cross-agent comparison, conflict resolution, and final answer formation. The runtime graph therefore contains four LLM-backed task roles and three directed edges from the specialists to the summarizer. The terminal role is optimized separately because its input is a concatenation of upstream responses rather than the original task alone.

Sequential topology. The Sequential topology forms an ordered chain of prediction and reflection stages. With the experimental setting $N_r = 2$, the execution is equivalent to two predictor–reflector pairs connected in sequence. Each reflector receives the preceding prediction and is instructed to identify errors, missing cases, or unsupported conclusions before the next predictor continues the solution. This topology exposes the effect of upstream prompt and model choices on all later stages, while also making the reflector capability important for error detection.

Decentralized topology. The Decentralized topology performs two rounds of peer interaction. Three logical agents first produce independent analyses; in the second round, the corresponding agents revisit the task after observing the other agents' outputs. The two runtime instances associated with the same logical role share its prompt and model assignment, giving six LLM-backed calls but only three logical assignment variables. A mechanical voter combines the final peer outputs and is excluded from the deployment-time LLM cost.

Dynamic topology: context-conditioned role selection. The Dynamic topology uses a pool of five dataset-specific roles rather than a fixed static execution order. The role pool is specialized to the task: it contains reasoning, analysis, verification, and finalization roles for AGIEval-MATH and GPQA; intent, function matching, validation, and refinement roles for BFCL-V3; interpretation, algorithm design, implementation, and code review roles for LiveCodeBench; and entity analysis, evidence collection, bridge reasoning, and answer verification roles for HotpotQA. Each dataset also has a dedicated final-answer role.

At each non-final step, a selector observes the task, the accumulated role outputs, the available role descriptions, and the current execution history. It then chooses the next role from the candidate pool. The implementation allows a role to be selected more than once and allows other roles to be skipped, subject to a maximum depth of five and a forced final-answer slot. This makes the realized computation path input-dependent: two instances from the same benchmark may use different subsets or orders of roles.

This design follows the dynamic speaker-selection pattern exemplified by AutoGen's `SelectorGroupChat`, in which an LLM selects the next agent from role descriptions and shared conversation history rather than following a fixed round-robin schedule (Wu et al., 2024)[1] Our implementation adapts that idea to a bounded role pipeline: the selector chooses among task-specific roles, while the selected role performs the actual task reasoning and emits a structured intermediate result.

The selector is a control component and is distinct from the task-agent model allocation optimized by UniMASO. In the reported implementation it uses a fixed Qwen3.5-9B selector model; selected task roles may use any model from the five-model pool. The final role is responsible for satisfying the dataset-specific output contract, such as choice tags, canonical function-call JSON, executable code, or a short answer.

[1] AutoGen `SelectorGroupChat` documentation: `https://microsoft.github.io/autogen/dev/user-guide/agentchat-user-guide/selector-group-chat.html`.

Table 7: Performance comparison of UniMASO against homogeneous model assignments across five execution topologies. For each topology, we compare five homogeneous configurations with the heterogeneous configuration selected by UniMASO, reporting both task quality and deployment cost.

Topology	Model	AGIEval		GPQA		BFCL		LiveCodeBench		HotpotQA		Avg.	
		Acc.	Cost	Acc.	Cost	Acc.	Cost	p@1	Cost	EM	Cost	Result	Cost
Independent	Qwen3.5-4B	62.11	4.9	66.16	4.6	86.60	2.5	19.08	2.8	57.80	3.6	58.35	3.68
	Qwen3.5-9B	59.38	7.6	68.69	8.5	87.80	6.6	23.66	5.7	57.60	7.0	59.43	7.07
	Qwen3.5-27B	69.92	35.9	68.69	34.1	93.40	23.1	29.01	21.6	59.60	30.9	64.12	29.12
	DeepSeek-V4-Flash	67.19	43.9	69.70	27.1	93.00	28.1	36.64	29.4	56.40	59.0	64.59	37.50
	DeepSeek-V4-Pro	69.53	140.8	70.20	83.8	93.40	83.7	40.46	68.9	58.20	178.8	66.36	111.20
	UniMASO	73.44	30.9	73.74	24.5	93.00	11.4	35.11	22.3	59.00	20.0	66.86	21.81
Centralized	Qwen3.5-4B	62.11	7.3	67.17	5.3	88.40	4.8	18.32	3.8	58.00	5.8	58.80	5.38
	Qwen3.5-9B	71.09	14.3	71.72	12.2	90.40	9.1	22.14	8.1	56.80	10.5	62.43	10.84
	Qwen3.5-27B	75.78	56.2	77.27	41.0	90.40	38.7	29.77	26.4	60.80	48.5	66.80	42.18
	DeepSeek-V4-Flash	72.27	57.3	70.20	24.8	92.20	73.1	38.17	39.6	59.20	100.2	66.41	59.00
	DeepSeek-V4-Pro	76.17	217.3	78.28	103.4	92.00	195.1	32.82	112.0	61.60	295.0	68.17	184.55
	UniMASO	82.42	56.86	78.28	33.02	92.40	12.21	35.11	13.27	63.80	39.26	70.40	30.92
Sequential	Qwen3.5-4B	61.33	3.7	52.02	6.6	72.60	4.0	27.48	3.3	52.00	4.7	53.09	4.45
	Qwen3.5-9B	69.92	8.4	64.14	12.4	91.80	9.2	25.95	6.6	52.20	8.6	60.80	9.03
	Qwen3.5-27B	81.64	35.0	58.59	47.9	89.40	35.7	32.06	26.5	57.00	38.3	63.74	36.69
	DeepSeek-V4-Flash	75.39	27.5	68.18	16.8	91.00	40.4	34.35	36.5	55.20	52.9	64.82	34.82
	DeepSeek-V4-Pro	72.27	165.2	74.75	110.9	90.00	121.4	38.17	77.3	59.00	248.8	66.84	144.72
	UniMASO	81.64	51.1	76.26	47.7	93.20	31.3	38.17	31.2	50.20	54.9	67.89	43.23
Decentralized	Qwen3.5-4B	44.92	8.5	45.96	2.6	83.00	3.7	21.37	4.9	50.40	7.4	49.13	5.42
	Qwen3.5-9B	64.45	24.6	49.49	22.1	89.60	7.3	25.19	10.6	55.40	19.4	56.83	16.78
	Qwen3.5-27B	70.70	97.4	59.60	89.8	90.60	29.5	33.59	44.1	57.40	113.8	62.38	74.93
	DeepSeek-V4-Flash	77.73	120.6	69.19	103.9	93.40	72.9	32.82	86.7	54.00	123.9	65.43	101.62
	DeepSeek-V4-Pro	76.17	277.2	69.19	150.5	91.20	220.2	40.46	155.3	57.80	395.9	66.96	239.80
	UniMASO	72.27	73.8	77.27	87.4	92.00	23.3	34.35	56.3	57.00	61.0	66.58	60.37
Dynamic	Qwen3.5-4B	74.22	2.3	73.74	5.1	88.00	2.7	29.01	3.0	62.60	6.3	65.51	3.88
	Qwen3.5-9B	75.39	4.0	76.26	10.6	93.00	5.1	30.53	6.4	60.00	11.8	67.04	7.61
	Qwen3.5-27B	76.17	17.1	79.80	40.4	92.00	20.5	38.17	24.6	59.80	54.7	69.19	31.46
	DeepSeek-V4-Flash	78.12	19.2	78.28	41.8	93.00	40.0	38.93	36.0	59.40	107.5	69.55	48.91
	DeepSeek-V4-Pro	76.56	56.1	79.80	109.8	92.80	120.0	40.46	76.9	60.80	307.8	70.08	134.13
	UniMASO	81.25	16.0	80.30	24.6	92.00	7.6	39.69	20.7	58.60	34.0	70.37	20.56

Table 7 gives the per-task scores and deployment costs underlying Figure 3.

E Metrics and Cost Accounting

We use accuracy for AGIEval-MATH, GPQA-Diamond, and BFCL-V3, Pass@1 for LiveCodeBench, and exact match for HotpotQA. The aggregate score is their unweighted mean. Deployment cost is the token-weighted inference cost of task agents during final execution. Optimization overhead, evaluator calls, Co-Evolver calls, and Dynamic-selector calls are excluded from this metric.

E.1 Relative Cost Model

The cost coefficients are relative token-price weights rather than monetary prices. We construct the relative scale with reference to the per-token model prices listed by the Qwen AI model platform[2] and keep the resulting weights fixed across all experiments. The coefficients should therefore be interpreted as a reproducible cost ranking and not as a claim about the exact currency amount charged by a particular provider or account.

For a task execution containing calls $i = 1, \ldots, n$, we compute

$$\mathrm{Cost}(z) = \sum_{i=1}^{n} \left(c^{\mathrm{in}}_{m_i} T^{\mathrm{in}}_i + c^{\mathrm{out}}_{m_i} T^{\mathrm{out}}_i \right), \tag{9}$$

where m_i is the model used by call i, T^{in}_i and T^{out}_i are the measured input and output token counts, and c^{in}_m and c^{out}_m are the normalized model-specific coefficients. The values used by the implementation are shown in Table 8.

[2]Qwen AI model pricing reference: `https://www.qianwenai.com/models`.

Table 8: Model capability levels and relative token-cost coefficients.

Model	Capability	c^{in}	c^{out}
Qwen3.5-4B	Very weak	1	4
Qwen3.5-9B	Medium	2	8
Qwen3.5-27B	Strong	8	32
DeepSeek-V4-Flash	Very strong	24	72
DeepSeek-V4-Pro	Extremely strong	72	216

Relative to Qwen3.5-9B, the total reference weights are $0.5\times$, $1\times$, $4\times$, $9.6\times$, and $28.8\times$, respectively. These relative values enter cost-aware candidate selection through the cost weight λ; they do not alter the task-quality metrics.

F MAS PROMPT INITIALIZATION

This appendix lists the initial prompts used before UniMASO optimization. We first give the centralized topology, whose execution graph is $0 \to 3$, $1 \to 3$, and $2 \to 3$: agents 0–2 receive the input independently, while agent 3 receives their serialized outputs through $\{\texttt{context}\}$ together with the original input through $\{\texttt{question}\}$. The four prompt boxes for each dataset are transcribed from the corresponding runtime initialization templates. Following the MASPO layout, each box identifies its role in the gray title bar.

F.1 CENTRALIZED TOPOLOGY

F.1.1 AGIEVAL-MATH

AGIEval-MATH uses the centralized choice-task initialization. Agents 0–2 provide independent domain-oriented solutions, and agent 3 summarizes their outputs into the final choice.

Predictor Prompt (Math)

```
You are a math agent. Given the input question, reason step-by-step
and put the final answer inside <answer>YOUR_FINAL_CHOICE</answer>.
Your final answer must be selected from A,B,C,D. For example
<answer>A</answer>. Do not add any other contents inside the box.

Input Question: {question}

Your response:
```

Predictor Prompt (Science)

```
You are a science agent. Given the input question, reason
step-by-step and put the final answer inside
<answer>YOUR_FINAL_CHOICE</answer>. Your final answer must be
selected from A,B,C,D. For example <answer>A</answer>. Do not add
any other contents inside the box.

Input Question: {question}

Your response:
```

Predictor Prompt (Code)

```
You are a code agent. Given the input question, reason step-by-step
and put the final answer inside <answer>YOUR_FINAL_CHOICE</answer>.
Your final answer must be selected from A,B,C,D. For example
<answer>A</answer>. Do not add any other contents inside the box.

Input Question: {question}

Your response:
```

Summarizer Prompt

```
You are a task summarizer. Given the final answer inside
<answer>YOUR_FINAL_CHOICE</answer>. Your final answer must be
selected from A,B,C,D. For example <answer>A</answer>. Do not add
any other contents inside the box.

Content from Previous Agent:
{context}

Input Question: {question}

Your response:
```

F.1.2 GPQA-DIAMOND

GPQA-Diamond uses the same centralized choice-task prompt family, with the question-specific roles interpreted as mathematical, scientific, coding, and terminal-summarization specialists.

Predictor Prompt (Math)

```
You are a math agent. Given the input question, reason step-by-step
and put the final answer inside <answer>YOUR_FINAL_CHOICE</answer>.
Your final answer must be selected from A,B,C,D. For example
<answer>A</answer>. Do not add any other contents inside the box.

Input Question: {question}

Your response:
```

Predictor Prompt (Science)

```
You are a science agent. Given the input question, reason
step-by-step and put the final answer inside
<answer>YOUR_FINAL_CHOICE</answer>. Your final answer must be
selected from A,B,C,D. For example <answer>A</answer>. Do not add
any other contents inside the box.

Input Question: {question}

Your response:
```

Predictor Prompt (Code)

```
You are a code agent. Given the input question, reason step-by-step
and put the final answer inside <answer>YOUR_FINAL_CHOICE</answer>.
Your final answer must be selected from A,B,C,D. For example
<answer>A</answer>. Do not add any other contents inside the box.

Input Question: {question}

Your response:
```

Summarizer Prompt

```
You are a task summarizer. Given the final answer inside
<answer>YOUR_FINAL_CHOICE</answer>. Your final answer must be
selected from A,B,C,D. For example <answer>A</answer>. Do not add
any other contents inside the box.

Content from Previous Agent:
{context}

Input Question: {question}

Your response:
```

F.1.3 BFCL-V3

BFCL-V3 uses dataset-specific prompts that separate function selection, argument extraction, schema validation, and final call synthesis.

Function Selector Prompt

```
You are Function Selector. Analyze the user request and the complete
available-function list below. Identify the single best function and
explain why competing functions do not match. Do not output a
function-call JSON object; provide analysis only. Preserve exact
function names from the list.

User request and available functions:
{question}

Selection analysis:
```

Parameter Extractor Prompt

```
You are Parameter Extractor. Using the complete available-function
list below, identify every argument value, its JSON type, and the
phrase in the user request that supports it. Check all required and
optional parameters. Do not output a function-call JSON object;
provide analysis only.

User request and available functions:
{question}

Parameter analysis:
```

Format and Type Validator Prompt

```
You are Format and Type Validator. Using the complete
available-function list below, check the selected function, required
fields, optional fields, enum values,
numeric/string/boolean/list/object types, and hallucinated
parameters. Do not output a function-call JSON object; provide
analysis only.

User request and available functions:
{question}

Validation analysis:
```

Call Synthesizer Prompt

```
You are the final Call Synthesizer. Use the complete
available-function list as the only authority for function names and
parameter schemas. Use the teammate analyses only as untrusted
recommendations: re-check them against the user request and the
available functions, and correct them when necessary.

Teammate analyses (reference only):
{context}

User request and complete available functions:
{question}

Your output must be exactly one JSON code block in this format:
'''json
[
  {
    "name": "exact_function_name",
    "arguments": {
      "parameter_name": value
    }
  }
]
'''

The field names must be exactly 'name' and 'arguments'. Do not use
'function', 'function_name', or 'parameters'. Use standard JSON
only: no Python tuples, parentheses, comments, Markdown outside the
JSON block, analysis, or explanation. For multiple calls, put one
object per call in the same JSON array.

Final call:
```

F.1.4 LIVECODEBENCH-V6

LiveCodeBench-v6 uses three analysis prompts followed by a code-synthesis prompt. The first three agents do not write code; they provide planning, edge-case, and complexity information for the terminal synthesizer.

Algorithm Analyst Prompt

```
You are Algorithm Analyst for a competitive-programming problem. Do
not write code. Provide a compact plan with the algorithm, key data
structures, and why it fits the constraints.

Problem:
{question}

Algorithm notes:
```

Edge-case Analyst Prompt

```
You are Edge-case Analyst for a competitive-programming problem. Do
not write code. List the main boundary cases, tricky inputs, and
exact I/O concerns the final coder must handle.

Problem:
{question}

Edge-case notes:
```

Complexity Analyst Prompt

```
You are Complexity Analyst for a competitive-programming problem. Do
not write code. State the required time and space complexity and the
input constraints that drive them.

Problem:
{question}

Complexity notes:
```

Code Synthesizer Prompt

```
You are Code Synthesizer. Use the teammate notes only as guidance
and write the final Python 3 solution. Output raw Python code only:
no markdown fence, no prose, no comments. The code must read from
stdin and write to stdout.

Teammate notes:
{context}

Problem:
{question}

Final code:
```

F.1.5 HOTPOTQA

HotpotQA uses dataset-specific prompts for document analysis, evidence validation, multi-hop reasoning, and short-answer synthesis.

Document Analyst Prompt

```
You are Document Analyst. Extract relevant entities, dates, titles,
relationships, and facts from the provided documents. Do not guess
the final answer unless the evidence is explicit.

{context}

Question:
{question}

Relevant facts:
```

Evidence Validator Prompt

```
You are Evidence Validator. Verify which document facts support or
refute possible answers, and identify the strongest evidence chain.
Do not output a final answer tag.

{context}

Question:
{question}

Validated evidence:
```

Reasoning Specialist Prompt

```
You are Reasoning Specialist. Build a concise multi-hop reasoning
path that connects the question to the answer using document facts.
Do not output a final answer tag.

{context}

Question:
{question}

Reasoning path:
```

Answer Summarizer Prompt

```
You are Answer Summarizer. Synthesize the teammate analyses and
output only the final short answer. Use <answer>...</answer> tags
and put no reasoning inside the tags.

Teammate analyses:
{context}

Question:
{question}

Final answer:
```

The placeholders are filled at runtime. {question} denotes the current dataset item, while {context} denotes the predecessor outputs supplied to a downstream agent. The centralized

topology contains no separate mechanical aggregation prompt; agent 3 is the LLM-backed terminal role. The optimization-side prompts below are the templates used to evolve these initialized MAS prompts.

F.2 Optimization Prompts

The optimizer, evaluator, and Co-Evolver are separate LLM calls from the task agents. Their templates are selected by task type in the implementation. To avoid repeating identical scaffolding, the boxes below show the runtime protocols and the task-specific criteria that distinguish the variants. All placeholders are filled before the call; braces that must survive into a task prompt are escaped by the implementation.

F.2.1 Optimizer Prompts

The optimizer proposes a complete prompt–model candidate from execution traces, requirements, the current prompt, and the available model pool. The multiple-choice variant is shown first; the function-calling, code-generation, and QA variants retain the same interface while replacing the diagnostic criteria with the corresponding task contract.

Optimizer Prompt (Multiple Choice)

```
You are optimizing a prompt AND selecting a model for an agent in a
multiple-choice reasoning system.

Agent Type: {agent_type}
System Role: {role_description}
Current Model: {current_model}

============================== MODEL POOL
==============================
{model_pool_section}
================================================================================

{trajectory_section}

Sample Execution Traces (Question + Context + Agent Output)
```
{samples}
```
Requirements:
```
{requirements}
```
Reference prompt:
```
{prompt}
```

DECISION FRAMEWORK:
Step 1 - Diagnose output quality: What is wrong?
Step 2 - Assess model adequacy: Is {current_model} overkill,
adequate, or insufficient?
Step 3 - Output the optimized prompt and selected model.

Rules (cost-benefit balance, lambda={lambda_cost}):
- Optimize the prompt first. Then decide if model change is needed.
- Jump at most ONE tier per round, unless lambda=0.
- Upgrade when quality is insufficient; downgrade when quality is
excessive.
- Do NOT downgrade when prompt changes cannot fix clear errors.
```

```
OUTPUT (strict XML, nothing after </prompt>):
<analyse>Analysis of output quality and model adequacy.</analyse>
<model_decision>One of: Keep | Upgrade to <name> | Downgrade to
<name>. One sentence why.</model_decision>
<model>model_name_here</model>
<prompt>The complete optimized prompt. Preserve {{question}} and
{{context}} placeholders.</prompt>
```

Optimizer Prompt (Function Calling)

```
You are optimizing a prompt AND selecting a model for an agent in a
tool-calling / function selection system.

Agent Type: {agent_type}
System Role: {role_description}
Current Model: {current_model}

MODEL POOL:
{model_pool_section}

{trajectory_section}

Sample Execution Traces (User Query + Context + Agent Output)
```
{samples}
```
Requirements:
```
{requirements}
```
Reference prompt:
```
{prompt}
```

DECISION FRAMEWORK:
1. Diagnose wrong function names, missing required parameters,
incorrect types,
   hallucinated values, or invalid JSON fields.
2. Assess whether {current_model} is overkill, adequate, or
insufficient.
3. Optimize the prompt first, then decide whether a one-tier model
change is justified.

OUTPUT (strict XML, nothing after </prompt>):
<analyse>Analysis of function-call quality and model
adequacy.</analyse>
<model_decision>One of: Keep | Upgrade to <name> | Downgrade to
<name>. One sentence why.</model_decision>
<model>model_name_here</model>
<prompt>The complete optimized prompt. Preserve {{question}} and
{{context}} placeholders.</prompt>
```

Optimizer Prompt (Code Generation)

```
You are optimizing a prompt AND selecting a model for an agent in a
competitive programming system.
```

```
Agent Type: {agent_type}
System Role: {role_description}
Current Model: {current_model}

MODEL POOL:
{model_pool_section}

{trajectory_section}

Sample Execution Traces (Problem + Context + Agent Output)
```
{samples}
```
Requirements:
```
{requirements}
```
Reference prompt:
```
{prompt}
```

DECISION FRAMEWORK:
1. Diagnose algorithm errors, wrong I/O, missing edge cases,
inefficiency, or formatting violations.
2. Assess model adequacy and whether the observed errors can be
fixed by prompting.
3. Fix the prompt first; upgrade or downgrade by at most one tier
when evidence supports it.

OUTPUT (strict XML, nothing after </prompt>):
<analyse>Analysis of code quality issues and model
adequacy.</analyse>
<model_decision>One of: Keep | Upgrade to <name> | Downgrade to
<name>. One sentence why.</model_decision>
<model>model_name_here</model>
<prompt>The complete optimized prompt. Preserve {{question}} and
{{context}} placeholders.</prompt>
```

Optimizer Prompt (Question Answering)

```
You are optimizing a prompt AND selecting a model for an agent in a
multi-hop question answering system.

Agent Type: {agent_type}
System Role: {role_description}
Current Model: {current_model}

MODEL POOL:
{model_pool_section}

{trajectory_section}

Sample Execution Traces (Documents + Question + Context + Agent
Output)
```
{samples}
```
Requirements:
```
```
```

```
{requirements}
```
Reference prompt:
```
{prompt}
```

DECISION FRAMEWORK:
1. Diagnose missing cross-document connections, incomplete evidence chains,
   hallucinated facts, or answers that are too long or off-target.
2. Assess whether {current_model} is overkill, adequate, or insufficient.
3. Optimize the prompt first, then make a cost-justified one-tier model decision.

OUTPUT (strict XML, nothing after </prompt>):
<analyse>Analysis of QA output quality and model adequacy.</analyse>
<model_decision>One of: Keep | Upgrade to <name> | Downgrade to <name>. One sentence why.</model_decision>
<model>model_name_here</model>
<prompt>The complete optimized prompt. Preserve {{question}} and {{context}} placeholders.</prompt>

F.2.2 EVALUATOR PROMPTS

The evaluator is used at three granularities: comparing the focal agent's outputs, comparing intermediate outputs that will be consumed by successors, and comparing final system answers. The task-specific evaluator variants replace the criteria (for example, function-schema validity, code correctness, or evidence grounding) but preserve the binary comparison protocol.

Evaluator Prompt (Task-Output Comparison)

You are evaluating two outputs (A and B) from an agent of type {agent_type}.
Based on the input, requirements and the agent's role, determine which response is better.

The agent's specific role: {role_description}

INPUT:
{question}

REQUIREMENT:
{requirement}

OUTPUT A:
{Answer_A}

OUTPUT B:
{Answer_B}

GUIDELINES:
- Prefer a clearly correct answer over a longer or more confident answer.
- For multiple choice, check the single answer marker.
- For code, check syntax, algorithmic correctness, I/O, and edge cases.
- For QA, check evidence grounding and penalize fabricated facts.

```
- For function calling, check the exact function name, required
parameters, types,
  and the JSON array with name/arguments fields.

Provide analysis and the choice using XML tags:
<analyse>Comparison of correctness and task-specific
validity.</analyse>
<choose>A/B</choose>
```

Evaluator Prompt (Intermediate Comparison)

```
You are comparing two intermediate outputs from an agent in a
multi-agent reasoning system.
Your task is to decide which output is more likely to help the
system eventually produce the CORRECT FINAL ANSWER.

Prefer the output that:
- Is complete rather than truncated.
- Contains accurate reasoning and no factual or logical errors.
- Clearly states intermediate results or assumptions.
- Avoids misleading statements or ambiguous conclusions.
- Provides enough detail for downstream agents to verify or build
upon.

Problem: {question}
Output A:
{output_a}
Output B:
{output_b}

Which output is more conducive to obtaining the correct final
answer? Respond ONLY with "A" or "B".
```

Evaluator Prompt (Final Answer Comparison)

```
You are comparing two FINAL answers to a problem.
Your SOLE task is to determine which answer is more likely to be
correct.

Do NOT favor longer, more detailed, or better-formatted responses
unless they are also correct.
If one answer is clearly correct and the other is wrong, choose the
correct one.
If both seem plausible, prefer the one with clearer, error-free
reasoning.

Problem: {question}
Requirement: {requirement}

Answer A:
{Answer_A}

Answer B:
{Answer_B}

Respond with only "A" or "B".
```

F.2.3 CO-EVOLUTION PROMPTS

At the end of an optimization round, each focal view invokes the Co-Evolver. The static version receives the complete execution graph, while the dynamic version receives the role pool and excludes the selector from model assignment. Both versions return a final prompt and a feasible model assignment as JSON; the implementation then merges the proposals subject to topology constraints.

Co-Evolver Prompt (Static Topology)

```
You are the Co-Evolver at the end of one optimization round in a
heterogeneous multi-agent system.
Use the available topology, role, prompt, trajectory, and model-pool
evidence to construct a topology-aware co-evolution result.
Agent {agent_id} is the local perspective, but model_assignment must
cover all optimizable logical LLM agents and exclude mechanical
aggregation or voting operations.

GLOBAL TOPOLOGY AND ROLE INFORMATION
{topology_description}

AVAILABLE MODEL POOL
{model_pool_section}

CURRENT AGENT PROMPT EVOLUTION
Initial prompt at the start of this round:
{initial_prompt}
Best prompt after depth optimization:
{best_prompt}
Initial model: {initial_model}
Best model after depth optimization: {best_model}

RETRIEVED PROMPT-SIMILAR TRAJECTORIES
{experience_context}

TASK
1. Produce a final prompt for the current Agent. Preserve its role,
   placeholders, and required answer format; retain only valuable
improvements.
2. Produce a complete model_assignment for all optimizable logical
agents.
   Consider topology, role, dependencies, task difficulty,
critical-path
   responsibility, capability, and input/output cost.
3. Do not use historical candidate prompts or trajectory scores to
determine
   model assignment. Do not assign models to mechanical nodes.

Return exactly:
{
  "final_prompt": "one prompt string",
  "model_assignment": {"agent_id": "model_name"},
  "decision_summary": "short factual summary"
}
```

Co-Evolver Prompt (Dynamic Topology)

```
You are the Co-Evolver at the end of one optimization round in a
heterogeneous multi-agent system.
Use the available role-pool, prompt, trajectory, and model-pool
evidence to construct a role-aware co-evolution result.
```

```
Agent {agent_id} is the local perspective, but model_assignment must
cover all five optimizable logical LLM roles and must exclude the
Dynamic Selector.

DYNAMIC ROLE POOL INFORMATION
Dataset: {dataset}
Role pool:
{role_pool}
Selector history and realized route:
{route_history}

CURRENT ROLE PROMPT EVOLUTION
Initial prompt:
{initial_prompt}
Best prompt after depth optimization:
{best_prompt}
Initial model: {initial_model}
Best model: {best_model}

TASK
1. Produce a concise final prompt for the current role. Preserve all
   placeholders, delimiters, and dataset-required output structures.
2. Produce a complete model_assignment for the five roles using only
the
   available model pool. Consider role responsibility, route
frequency,
   critical-path impact, task difficulty, capability, and cost.
3. Do not assign a model to the Selector and return only JSON.

Return exactly:
{
  "final_prompt": "one prompt string",
  "model_assignment": {"role_id": "model_name"},
  "decision_summary": "short factual summary"
}
```