# AgentBalance: Budget-Aware Co-Design for Heterogeneous LLM Multi-Agent Systems

Anonymous ACL submission


## Abstract

Large language model (LLM)-based Multi-Agent Systems (MAS) improve problem solving by coordinating specialized agents, but their practical deployment is often constrained by token-cost and end-to-end latency. Existing approaches have explored backbone routing and communication optimization, but often emphasize one side of the design space or leave their coupling implicit, even though each interaction's utility and latency cost depend on the assigned backbones in heterogeneous systems. In this paper, we introduce AGENTBALANCE, a budget-aware co-design framework for heterogeneous LLM multi-agent systems. Given token-cost and latency budgets, AGENTBALANCE co-designs role-specific backbone assignments and backbone-conditioned communication structures through a staged backbone-then-topology procedure: it first constructs budget-aligned backbone pools and performs query- and role-aware model assignment, then synthesizes sparse latency-aware topologies conditioned on the instantiated agents. Experiments across knowledge, reasoning, and coding benchmarks, together with database-querying plug-in evaluations, show that AGENTBALANCE consistently improves the performance–token-cost–latency frontier over baselines. Further analyses demonstrate its effectiveness as a plug-in configurator for existing MAS and its adaptability to unseen LLM backbones.


## 1 Introduction

Large Language Model (LLM)-based multi-agent systems (MAS) have emerged as a powerful paradigm for solving complex tasks by decomposing problems across multiple specialized agents and coordinating their reasoning through inter-agent communication (Hong et al., 2024; Li et al., 2023a). By assigning different roles to agents and allowing them to exchange intermediate outputs, these systems have shown strong performance across reasoning, coding, information seeking, and tool-use scenarios (Schick et al., 2023; Chen et al., 2025; Rezaei et al., 2025). However, this performance often comes with substantial deployment overhead: each agent invocation increases token-cost, while deeper or denser communication structures increase end-to-end latency. As LLM-based MAS move from experimental settings toward practical applications, token-cost and latency constraints become central design requirements rather than secondary efficiency concerns (Ning et al., 2025; Embar et al., 2025; Zhao et al., 2023). This motivates budget-aware MAS design that improves task performance while respecting token-cost and latency budgets, pushing the performance–token-cost–latency frontier, as illustrated in Figure 1.

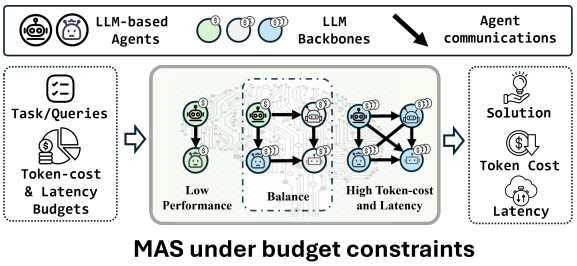



Figure 1: Multi-agent System Under budget Constraints. AGENTBALANCE aims to maximize task performance under given token-cost and latency budgets.

A growing body of work has therefore studied cost-effective multi-agent systems. One line of research reduces the expense of LLM agents by selecting or routing among different backbone models (Ma et al., 2025; Yue et al., 2025; Yang et al., 2025b), while another line improves efficiency by pruning agents or optimizing communication topologies (Zhang et al., 2025b,c). While these methods have made important progress, they typically emphasize either backbone assignment or communication structure, leaving the coupling between them under budgets only implicitly modeled. The utility of a communication edge is not fixed: it depends on the capabilities, token-costs, latencies, and reasoning behaviors of the backbones

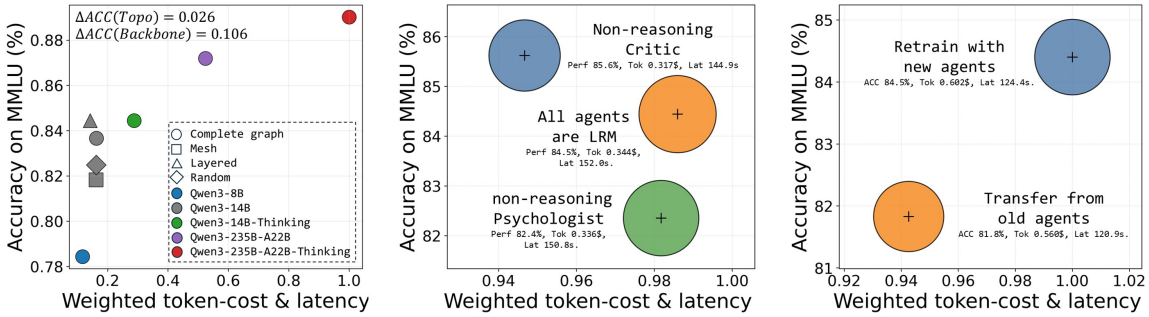



Figure 2: **(Left)** Backbone choice strongly shapes the MAS performance–budget frontier, with a larger frontier shift than topology-only designs. **(Middle)** With topology fixed, assigning a non-reasoning model (vs. an LRM) to a suitable role (*e.g.*, *Critic*) improves cost-effectiveness, whereas role-backbone mismatches (*e.g.*, *Psychologist*) degrade it. **(Right)** The optimal communication topology is backbone-dependent; after changing agent backbones, re-synthesizing the topology for the new agents outperforms transferring the old one.

assigned to the interacting agents. For example, a verification edge may be valuable when linked to a strong reasoning model, but wasteful when both agents use weak or redundant backbones; conversely, a lightweight model may be sufficient for critique or filtering roles even when stronger models are needed for generation. Thus, we argue cost-effective MAS construction requires reasoning about backbone assignment and communication structure design jointly.

However, realizing this co-design is challenging for three reasons. First, candidate backbones differ in accuracy, price, response length, latency, vendor, family, and reasoning mode. Such heterogeneity strongly shapes the MAS performance–budget frontier as shown in Figure 2-left, making simple "stronger-is-better" strategy ineffective under explicit budgets. Second, backbone suitability is role- and query-dependent: Large Reasoning Models (LRM) may be needed for difficult solution generation, while lightweight models may suffice for verification, critique, or filtering. Role–backbone mismatches can waste budgets or degrade performance as shown in Figure 2-middle. Third, communication topology interacts with backbone assignment, since the marginal benefit and latency overhead of each edge depend on the backbones assigned to the instantiated endpoint agents. A topology effective under one backbone assignment may become suboptimal after backbones change, as shown in Figure 2-right. These challenges motivate a staged co-design strategy: first constructing budget-aligned agents, then synthesizing communication structures conditioned on them.

To address these challenges, we propose **AGENTBALANCE**, a budget-aware co-design framework for heterogeneous LLM-based MAS under token-cost and latency budgets. AGENTBALANCE first performs *Backbone-Oriented Agent Generation*, which manages heterogeneous candidate LLMs, selects a budget-aligned LLM pool, and matches heterogeneous backbones to agent roles. It then conducts *Adaptive MAS Topology Generation*, which learns query- and backbone-aware agent representations, prunes redundant agents, and synthesizes a latency-aware communication topology. The whole framework is optimized end-to-end with token-cost and latency penalties to balance task performance and budget. Our contributions are:

1. **Budget-aware MAS problem.** We study cost-effective MAS construction under explicit token-cost and latency budgets, and formulate it as a coupled configuration problem over backbone assignment and communication structure design.
2. **Co-design framework.** We propose AGENTBALANCE, a budget-aware co-design framework for heterogeneous LLM-based MAS that first construct budget-aligned agents with backbone-oriented agent generation, then apply adaptive MAS topology generation to form cost-effective MAS.
3. **Comprehensive validation.** Experiments with 23 heterogeneous LLM backbones across three benchmarks show that AGENTBALANCE consistently improves performance under matched token-cost and latency budgets, while also serving as a plug-in for existing MAS and adapting to unseen LLMs.

## 2 Background and Problem Statement

**LLM-based Multi-agent Systems.** We model an LLM-based multi-agent system (MAS) as a directed acyclic graph (DAG) $G = (V, E)$, where nodes are agents and edges denote inter-agent

message passing. Each agent is represented as $v_i = \langle b_i, \rho_i, s_i, \omega_i \rangle \in V$, where $b_i \in \mathcal{B}$ is the assigned LLM backbone, $\rho_i$ is the role-specific prompt, $s_i$ is the runtime context state, and $\omega_i$ denotes optional tools or APIs. The input prompt $P_i$ of agent $v_i$ is assembled from its role prompt, runtime state, and messages on incoming edges, where $E_{ji} = 1$ indicates that a message is transmitted from $v_j$ to $v_i$. The agent then produces a response $\mathcal{R}_i = b_i(P_i)$, incurring agent-level token-cost $\mathrm{Tok}(b_i, P_i)$ and latency $\mathrm{Lat}(b_i, P_i)$. These agent-level outcomes depend strongly on the assigned backbone $b_i$: response quality, token-cost, and latency are shaped by model size, family, reasoning type, provider pricing, expected response length, and vendor-specific serving conditions. For example, Large Reasoning Models (LRMs) often improve reasoning by generating longer completions but increase token-cost and latency. Accordingly, the MAS token-cost is the sum of agent-level token-costs, while its end-to-end latency is the critical-path time of the DAG, *i.e.*, the longest accumulated execution time along any directed dependency chain.

By defining a MAS template for instantiate cost-effective MAS, we then formalize the budget-constrained MAS construction problem.

**Definition 1** (MAS Template)**.** *A MAS template is defined as $M = (G^*, \mathcal{B})$. Here, the template graph $G^* = (T, E^*)$ specifies a library of* agent templates *$T$ and the admissible edges $E^*$ among them. Each template $\tau = \langle \rho, \omega \rangle$ fixes the role instruction $\rho$ and optional tools $\omega$, while leaving the backbone unspecified. $\mathcal{B}$ denotes a set of heterogeneous LLM backbones $b$ available for MAS instantiation.*

Based on a MAS template, we can instantiate diverse MAS instances and adapt them to tasks and queries in a cost-effective manner under different budget requirements.

**Problem 1** (Budget-Constrained MAS Construction)**.** *For each query $Q \in \mathcal{D}$ , the configurator $F_\theta$ instantiates a query-specific MAS by selecting a subset of templates $T_Q \subseteq T$, assigning backbones to them to construct agents $V_Q$ via a mapping $\phi_Q : T_Q \to \mathcal{B}$, and choosing edges $E_Q \subseteq E^*$ to define communications between them:*

$$\begin{aligned} F_\theta(M, Q) \;&=\; (V_Q, E_Q), \\ V_Q \;&=\; \big\{ \langle \phi_Q(\tau),\, \tau.\rho,\, \tau.\omega \rangle \,:\, \tau \in T_Q \big\}. \end{aligned} \tag{1}$$

*Our goal is to learn $F_\theta$ to maximize task performance while keeping the* expected *token-cost and latency within explicit token-cost budget $B_{\mathrm{tok}}$ and latency budgets $B_{\mathrm{lat}}$:*

$$\begin{aligned} \max_{\theta} \quad & \mathbb{E}_{Q\sim\mathcal{D}}\big[\mathrm{Perf}\big(F_\theta(M,Q), Q\big)\big] \\ \textit{s.t.} \quad & \mathbb{E}_{Q\sim\mathcal{D}}\big[\mathrm{Tok}\big(F_\theta(M,Q), Q\big)\big] \le B_{\mathrm{tok}}, \\ & \mathbb{E}_{Q\sim\mathcal{D}}\big[\mathrm{Lat}\big(F_\theta(M,Q), Q\big)\big] \le B_{\mathrm{lat}}. \end{aligned} \tag{2}$$

*Here* $\mathrm{Perf}(\cdot)$, $\mathrm{Tok}(\cdot)$, $\mathrm{Lat}(\cdot)$ *denote the task performance, token-cost and end-to-end latency of MAS.*

## 3 Methodology

Figure 3 gives an overview of AGENTBALANCE. Given a query $Q$, AGENTBALANCE (a) first performs *backbone-oriented agent generation* to instantiate budget-aligned heterogeneous agents $V_Q$, and then (b) applies *adaptive MAS topology generation* to synthesize a latency-aware topology $E_Q$, forming $G_Q = (V_Q, E_Q)$. The whole configurator is optimized end-to-end under a budget-aware objective; the full workflow is shown in Algorithm 1.

### 3.1 Backbone-Oriented Agent Generation

Backbone-oriented agent generation constructs a budget-aligned set of candidate agents $V_Q$ in three steps: (i) *Pool Construction with Profiling*, which groups candidate backbones into resource pools; (ii) *Difficulty-Aware Pool Selection*, which selects a budget-aligned pool for query $Q$ with probability $p_{\mathrm{sel}}$; and (iii) *Query-Conditioned Role–Backbone Matching*, which assigns backbones from the selected pool to agent roles with probability $p_{\mathrm{match}}$.

#### 3.1.1 Pool Construction with Profiling

Candidate backbones are highly heterogeneous in capability, token-cost, latency, vendor, family, and reasoning type. To obtain comparable resource tiers, we first profile each backbone $b \in \mathcal{B}$ with $z_b = \langle \mathrm{Perf}_b, \mathrm{TokCost}_b, \mathrm{Lat}_b \rangle$, where $\mathrm{Perf}_b$ is derived from benchmark performance, $\mathrm{TokCost}_b$ from per-token price and expected token usage, and $\mathrm{Lat}_b$ from calibrated inference latency. We retain non-dominated backbones on the 3D Pareto frontier (Miettinen, 1999) and cluster them into LLM pools. Each query selects one pool, ensuring a coherent budget tier for all instantiated agents.

We further construct three textual backbone profiles: performance for capability, per-token price for cost, and type for reasoning behavior. A lightweight $\mathrm{TextEncoder}$ maps them into $e_b^{\mathrm{perf}}, e_b^{\mathrm{ptp}}, e_b^{\mathrm{type}} \in \mathbb{R}^H$, which serve as stable backbone descriptors for difficulty-aware pool selection and query-conditioned role–backbone matching.

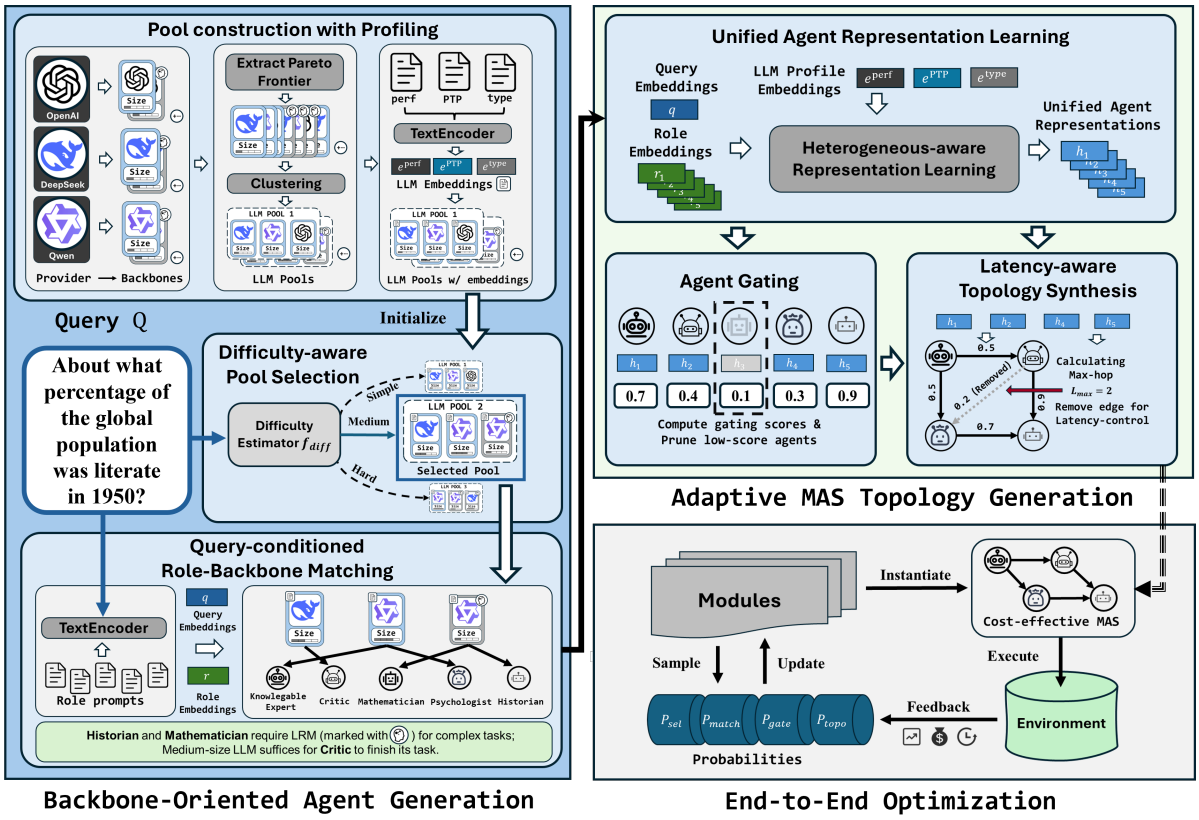



Figure 3: The overall framework of our proposed AGENTBALANCE

Details of calibration, clustering, and profile templates are provided in Appendix D.

### 3.1.2 Difficulty-Aware Pool Selection

To allocate the query-level budget, this module estimates the difficulty of query $Q$ and selects a corresponding LLM pool as the resource tier for subsequent role–backbone matching.

*Difficulty Estimation.* We train a lightweight difficulty estimator $f_{\text{diff}}$ on RouterBench (Hu et al., 2024), deriving supervision from the fraction of LLMs that solve each item. Given query $Q$, the estimator returns a normalized difficulty score $d = f_{\text{diff}}(Q) \in [0, 1]$. We further apply a difficulty offset $\delta$ to shift the operating point toward lower- or higher-budget regimes, and use a strict upper bound to prevent selecting pools above the allowed budget tier: $d_{\text{eff}} = d + \delta$.

*Pool Assignment.* LLM pools are ordered from weak to strong by aggregate performance. For each pool $p$, we average the embeddings of its member backbones to obtain pool representation $e_p$, and compute cost-aware logits

$$\ell_p = W_P e_p - \alpha \bar{c}_p,$$

where $\alpha$ is a learned cost-aversion coefficient and $\bar{c}_p$ is the normalized pool cost. After masking pools that exceed the upper bound, we use a masked softmax to form monotone thresholds over ordered pools, and apply a sigmoid-smoothed bucketizer $G$, following soft routing in differentiable decision trees (Kontschieder et al., 2015), to map $d_{\text{eff}}$ into a categorical pool distribution. The selected pool $\mathcal{P}_Q$ has probability

$$p_{\text{sel}} = G_{p^\star}\left(d_{\text{eff}}; \{\text{thr}_p\}_{p=1}^{P-1}\right).$$

Details of the pool selection are in Appendix F.

### 3.1.3 Query-Conditioned Role–Backbone Matching

Within the selected pool, we assign one backbone to each role by matching backbone profiles with query- and role-specific needs. For each backbone $b \in \mathcal{P}_Q$, we construct its representation by combining capability, cost, and reasoning-type descriptors:

$$u_b = W_u\left[e_b^{\text{perf}}; W_{\text{ct}}\left[e_b^{\text{ptp}}; e_b^{\text{type}}\right]\right].$$

On the role side, we encode the query and each role prompt with the TextEncoder to obtain the query embedding $q$ and role embedding $r_i$. To make matching aware of the selected resource tier, we compute a global pool context $g = W_g\left(\frac{1}{|\mathcal{P}_Q|}\sum_{b\in\mathcal{P}_Q}[e_b^{\text{perf}}; e_b^{\text{ptp}}; e_b^{\text{type}}]\right)$, and

construct the role representation $\psi_i = W_v[r_i; q; g]$. The backbone distribution for role $i$ is then $\pi_i(b) = \mathrm{softmax}_{b \in \mathcal{P}_Q}(\langle \psi_i, u_b \rangle)$, and the matching probability accumulated into the objective is:

$$p_{\text{match}} = \prod_{i=1}^{|T_Q|} \pi_i(b_i). \tag{3}$$

The selected backbones instantiate the heterogeneous agent set $V_Q$.

## 3.2 Adaptive MAS Topology Generation

Given heterogeneous agents $V_Q$, *Adaptive MAS Topology Generation* composes the final MAS $G_Q = (V_Q, E_Q)$ by synthesizing a latency-aware communication topology between them. It proceeds in three stages: (i) *Unified Agent Representation Learning* to obtain query- and backbone-aware representations; (ii) *Agent Gating* to remove redundant agents with $p_{\text{gate}}$; and (iii) *Latency-Aware Topology Synthesis* to generate a low-latency topology for MAS with $p_{\text{topo}}$.

### 3.2.1 Unified Agent Representation Learning

Agents produced by *backbone-oriented agent generation* are heterogeneous: their assigned backbones may differ in capability, cost, latency, family, and reasoning type, and such assignments also vary across queries. Since communication decisions should depend on both the agent role and its assigned backbone, we project each agent into a unified, query- and backbone-aware representation. This representation captures role semantics, query information, and backbone-specific capability/cost cues, and is later used by both agent gating and topology synthesis.

Specifically, for role $i$ with selected backbone $b_i$, we first form query and role embeddings $q^h = W_q q$ and $r_i^h = W_r r_i$. We then map the three backbone profile embeddings through a shared head as $d_i = W_\ell e_{b_i}^{\text{perf}}$, $c_i = W_\ell e_{b_i}^{\text{ptp}}$, and $t_i = W_\ell e_{b_i}^{\text{type}}$, corresponding to capability, cost, and reasoning-type information. Next, we compute $\mathbf{q}_i = W_Q[\, r_i^h; q^h \,]$, $\mathbf{k}_i = W_K[\, d_i; c_i; t_i \,]$, $\mathbf{o}_i = W_O[\, d_i; c_i; t_i \,]$, and get:

$$h_i = r_i^h + \gamma W_{\text{ctx}} \mathrm{attn}(\mathbf{q}_i, \mathbf{k}_i, \mathbf{o}_i), \tag{4}$$

where $\mathrm{attn}(\cdot)$ denotes scaled dot-product attention (Vaswani et al., 2017) and $\gamma$ is a trainable scale. This design injects a lightweight *query- and backbone-aware* context into the agent representation $\{h_i\}$, which facilitates *Agent Gating* and *Latency-Aware Topology Synthesis*.

### 3.2.2 Agent Gating

Not all agents are necessary for every query (*e.g.*, a *psychologist* for a math problem). Therefore, we prune redundant agents to preserve a compact and competent team. Given the unified agent representations $\{h_i\}_{i=1}^{|V_Q|}$, we first form a global context $\bar{h} = \frac{1}{|V_Q|} \sum_i h_i$, and compute the keep probability $p_i = \sigma(W_g[\, h_i; \bar{h} \,])$. We sample the binary gate $g_i \in \{0, 1\}$ with a Gumbel-Sigmoid relaxation (Louizos et al., 2018) during training, get

$$p_{\text{gate}} = \prod_{i=1}^{|V_Q|} p_i^{g_i} (1 - p_i)^{1 - g_i}. \tag{5}$$

If fewer than two agents are active, we retain the highest-$p_i$ agents. The retained set of agents $V_Q^{\text{gated}} = \{v_i \in V_Q : g_i = 1\}$ is then passed to the topology generator.

### 3.2.3 Latency-Aware Topology Synthesis

Over the retained agents, we synthesize a sparse communication DAG to control both communication overhead and end-to-end latency. Let $N = |V_Q^{\text{gated}}|$. For each ordered pair $i < j$, we compute the edge probability $p_{ij} = \sigma(\langle W_a h_i, W_a h_j \rangle)$, and sample $E_{ij}$ with a Gumbel-Sigmoid relaxation and the unified agent representations $\{h_i\}_{i=1}^{N}$ . We use the upper-triangular parameterization to enforce acyclicity of communication topology, and the topology probability is

$$p_{\text{topo}} = \prod_{1 \le i < j \le N} p_{ij}^{E_{ij}} (1 - p_{ij})^{1 - E_{ij}}. \tag{6}$$

Because larger communication depth directly increases end-to-end latency, we learn a query-conditioned hop limit $L_{\max}$ to constrain the longest-path length $\ell(E_Q)$. Given the retained agents, we summarize their representations by the pooled context $\bar{h} = \frac{1}{N} \sum_i h_i$, and use it to predict a categorical distribution $\pi_{\text{hop}} = \mathrm{softmax}(W_L \bar{h})$ over possible hop increments $k \in \{1, \ldots, N-1\}$. Intuitively, $\pi_{\text{hop}}(k)$ estimates how much additional sequential depth the current MAS can afford. We then set $L_{\max}$ as a one-hop baseline plus the expected hop increment:

$$L_{\max} = 1 + \sum_{k=1}^{N-1} k \, \pi_{\text{hop}}(k). \tag{7}$$

Let $\ell(E_Q)$ be the longest-path length on the sampled edges $E_Q$. If $\ell(E_Q) > L_{\max}$, we iteratively remove the lowest-probability edge on the current

critical path until the constraint is met. During training, we use $\mathrm{Pen}_{\mathrm{len}} = \mathrm{ReLU}(\ell(E_Q) - L_{\max})$ to encourage feasible low-latency topologies.

### 3.3 End-to-end Optimization

We optimize the configurator end-to-end with a Lagrangian surrogate of the budget-constrained objective in Eq. 2, trading off task performance, token-cost, and latency. The budgets specify deployment-time feasibility targets, while $\lambda_{\mathrm{tok}}$ and $\lambda_{\mathrm{lat}}$ are training-time penalty weights that control the operating point on the performance–budget frontier.

For a query $Q$, the instantiated MAS is $G_Q = F_\theta(M, Q)$, and the per-query reward is

$$\begin{aligned} R(G_Q, Q) = \mathrm{Perf}(G_Q, Q) \\ - \lambda_{\mathrm{tok}} \mathrm{Tok}(G_Q, Q) \\ - \lambda_{\mathrm{lat}} \mathrm{Lat}(G_Q, Q). \end{aligned} \tag{8}$$

$\lambda_{\mathrm{tok}}, \lambda_{\mathrm{lat}} \geq 0$ index operating points on the trade-off among performance, token-cost and latency (larger $\lambda$ encourages cheaper faster configurations).

Following our backbone-then-topology design, we decompose the joint configuration search into two conditioned stages: first restricting each query to a budget-aligned heterogeneous agent set, and then optimizing topology over the instantiated agents. This effectively reduces the search space, while hop-length control, upper-triangular DAG parameterization further stabilize training. For each query, the configurator samples a discrete decision tuple $\mathcal{D} = \{p^\star, \{b_i\}, \{g_i\}, E_Q\}$, covering pool selection, role–backbone matching, agent gating, and topology synthesis. We provide the constraint-handling rules used to reduce infeasible samples and stabilize the discrete search in Appendix G.

After executing $G_Q$ and obtaining the reward $R$, we aggregate the policy signal through the factorized decision probability:

$$p_\theta(\mathcal{D} \mid Q) = \underbrace{p_{\mathrm{sel}}}_{\text{pool selection}} \cdot \underbrace{p_{\mathrm{match}}}_{\text{matching}} \cdot \underbrace{p_{\mathrm{gate}}}_{\text{gating}} \cdot \underbrace{p_{\mathrm{topo}}}_{\text{topology}} \cdot \tag{9}$$

We then optimize a policy-gradient objective with a hop-length regularizer:

$$\begin{aligned} \mathcal{L} = & - R(G_Q, Q) \log p_\theta(\mathcal{D} \mid Q) \\ & + \lambda_{\mathrm{len}} \mathrm{Pen}_{\mathrm{len}}(E_Q, L_{\max}), \end{aligned} \tag{10}$$

where $\lambda_{\mathrm{len}} \geq 0$ balances hop-length regularization against reward. This objective increases the likelihood of high-reward MAS configurations while penalizing excessive communication depth, thereby balancing performance, token-cost, and latency. We follow standard practice in MAS design (Zhuge et al., 2024) and apply policy gradient (Williams, 1992) to minimize $\mathcal{L}$.

## 4 Experiments

### 4.1 Experimental Setup

**Benchmarks** We evaluate AgentBalance on four benchmarks: **(1) MMLU** (Hendrycks et al., 2021a) for general knowledge; **(2) MATH** (Hendrycks et al., 2021b) for mathematical problem reasoning; **(3) HumanEval** (Chen et al., 2021) for code generation These benchmarks span multiple domains and reasoning skills, providing a comprehensive assessment of cross-domain effectiveness. **(4) Bird** (Li et al., 2023b) is evaluated for complex real-world database-querying scenarios in inductive ability analysis.

**Baselines and LLM Backbones** We compare against cost-effective MAS baselines, including the *single-backbone* methods AgentPrune (Zhang et al., 2025b), AgentDropout (Wang et al., 2025b), and G-Designer (Zhang et al., 2025c), as well as the *multi-backbone* baseline MasRouter (Yue et al., 2025). All baselines are optimized on Complete Graph (Qian et al., 2025). Candidate LLMs consist of 23 backbone models from two vendors as well as local vllm models (Kwon et al., 2023) are selected to evaluate the cost-effectiveness of AgentBalance. To characterize attainable trade-offs, we construct **performance–budget envelopes** for all methods: for single-LLM baselines, we evaluate all candidate backbones and take the upper envelope as the performance—budget frontier. For MasRouter and AgentBalance, we sweep hyperparameters to span a range of token-cost and latency budgets.

**Implementation.** We set the learning rate to 0.1 and fix the LLM decoding temperature at 0. AgentBalance instantiates four LLM pools and employs an MPNet TextEncoder (Song et al., 2020). Unless otherwise specified, we report mean results over five independent runs.

**Metrics.** We use *Performance-at-Budget* (Buckley and Voorhees, 2000) for both token-cost and latency. Specifically, $P@T$ denotes task performance subject to a token-cost budget, while $P@L$ analogously denotes performance under a latency budget. We use *accuracy* for MMLU and MATH, and *Pass@1* for HumanEval. To summarize performance across a range of budgets, we additionally

Table 1: Performance comparison with baselines under token-cost and latency budgets ($P@T_{1..4}$, $P@L_{1..4}$; details in Table 10) and AUCs. All metrics are higher-is-better; the best per dataset in **bold**.

| Dataset | Method | Under Token-Cost Budgets | | | | | Under Latency Budgets | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $P@T_1$ | $P@T_2$ | $P@T_3$ | $P@T_4$ | $\text{AUC}_{tok}$ | $P@L_1$ | $P@L_2$ | $P@L_3$ | $P@L_4$ | $\text{AUC}_{lat}$ |
| MMLU | AgentPrune | 59.68 | 79.71 | 84.28 | 86.50 | 1.269 | 32.88 | 71.25 | 81.29 | 87.49 | 237.7 |
| | AgentDropout | 61.45 | 80.65 | 83.66 | 85.83 | 1.262 | 32.58 | 81.92 | 81.30 | 84.54 | 238.2 |
| | G-Designer | 61.80 | 78.35 | 83.89 | 86.35 | 1.261 | 60.08 | 74.34 | 82.48 | 86.55 | 243.2 |
| | MasRouter | 15.18 | 33.02 | 77.04 | 82.11 | 1.117 | 53.22 | 81.13 | 83.29 | 84.97 | 239.2 |
| | AgentBalance | **71.90** | **83.89** | **85.82** | **86.71** | **1.297** | **72.21** | **83.66** | **85.67** | **87.84** | **250.0** |
| HumanEval | AgentPrune | 86.56 | 88.86 | 90.53 | 94.64 | 1.868 | 46.82 | 86.42 | 90.95 | 91.10 | 470.6 |
| | AgentDropout | 87.23 | 90.74 | 92.10 | 95.01 | 1.876 | 48.13 | 88.95 | 91.07 | 91.85 | 471.0 |
| | G-Designer | 86.90 | 90.29 | 92.16 | **95.52** | **1.885** | 57.29 | 88.35 | 91.42 | 92.21 | 474.9 |
| | MasRouter | 12.68 | 41.39 | 82.77 | 90.86 | 1.724 | 11.45 | 21.47 | 77.89 | 86.89 | 440.9 |
| | AgentBalance | **87.94** | **91.35** | **92.74** | 95.46 | 1.880 | **60.11** | **89.13** | **92.20** | **94.39** | **476.5** |
| MATH | AgentPrune | 63.96 | 72.09 | 75.74 | 78.96 | 3.515 | 59.42 | 64.89 | 72.09 | 75.49 | 592.9 |
| | AgentDropout | 62.75 | 72.17 | 74.91 | 78.01 | 3.471 | 58.30 | 64.08 | 72.44 | 75.52 | 588.5 |
| | G-Designer | 63.05 | 71.26 | 74.80 | 77.43 | 3.450 | 59.00 | 64.44 | 71.34 | 74.26 | 581.9 |
| | MasRouter | 19.26 | 39.46 | 67.81 | 73.81 | 3.259 | 35.54 | 65.44 | 70.46 | 75.63 | 565.4 |
| | AgentBalance | **66.46** | **73.33** | **75.83** | **79.11** | **3.523** | **64.03** | **73.37** | **76.02** | **79.38** | **602.4** |

report the area under the performance–token-cost curve and the performance–latency curve (AUC).

More experiments and details of experiment setup are in Appendix B and Appendix C.

### 4.2 Main Results

Table 1 compares AgentBalance with baselines under four token-cost and latency budgets, denoted as $P@T_{1..4}$ and $P@L_{1..4}$, together with AUC over the performance–budget curves.

We verify that AgentBalance consistently improves the performance–budget frontier with **(1) Token-cost efficiency**: AgentBalance achieves higher performance under matched token-cost budgets, especially in low-budget regimes; for example, on MMLU, it improves $P@T_1$ to **71.90**, outperforming G-Designer (61.80) and AgentDropout (61.45). **(2) Latency efficiency**: it also performs strongly under end-to-end latency constraints, improving low-latency performance on MATH from 59.42 to **64.03**, while introducing only about 0.1s configurator overhead. **(3) Global cost-effectiveness**: beyond individual budget points, AgentBalance achieves higher AUCs across datasets, indicating better overall trade-offs among performance, token-cost, and latency. In contrast, single-backbone MAS baselines cannot assign heterogeneous backbone across agents. MasRouter lacks explicit modeling of heterogeneous backbone under different budget regimes, leading to weaker performance especially under tight budgets.

### 4.3 Plug-in to Existing MAS

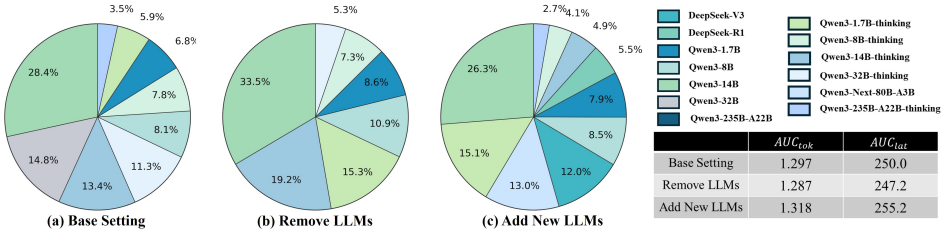
We evaluate AgentBalance as a framework-agnostic plug-and-play module by inserting it into existing MAS frameworks and replacing their backbone–topology configuration with AgentBalance. As shown in Table 2, AgentBalance consistently performs better under both token-cost and latency budgets. On MMLU, it brings substantial improvements to both AutoGen (Wu et al., 2024) and Layered Graph (Qian et al., 2025) across the specified budget points as well as in AUC. For Mac-SQL (Wang et al., 2025a), an industrial-grade data-processing MAS, AgentBalance likewise improves its performance throughout the full budget regime on *Bird* dataset. These results indicate that AgentBalance is able to serve as a cost-effective plug-in for existing MAS, fostering fast, practical and budget-aware deployment.

### 4.4 Inductive Ability Analysis

We evaluate whether AgentBalance generalizes to new LLM configurations without retraining. Two interventions are considered in evaluation: (i) two backbones (thinking and non-thinking versions of Qwen3-235B-A22B) are removed from the candidate backbones; (ii) three new backbones (Qwen3-Next-80B-A3B, DeepSeek-V3, DeepSeek-R1) are added. Figure 4 shows that AgentBalance reconfigures the MAS using the updated LLMs with even models not seen during training. When backbones are removed, the performance–budget frontier shifts downward and AUC decreases, yet the method remains superior to baselines. When new backbones are added, the frontier improves and AUC increases markedly, demonstrating strong inductive ability and rapid adaptation to new models without additional training resources.

Table 2: Plug-in effectiveness under token-cost and latency budgets ($P@T_{1..4}$, $P@L_{1..4}$; details in Table 11). AUCs of the performance–budget curves are also reported. All metrics are higher-is-better; the best per setting in **bold**.

| Method | Under Token-Cost Budgets | | | | | Under Latency Budgets | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $P@T_1$ | $P@T_2$ | $P@T_3$ | $P@T_4$ | $\text{AUC}_{tok}$ | $P@L_1$ | $P@L_2$ | $P@L_3$ | $P@L_4$ | $\text{AUC}_{lat}$ |
| Layered Graph (Qian et al., 2025) | 71.22 | 81.87 | 85.04 | 86.49 | 1.414 | 40.03 | 74.84 | 86.51 | 87.38 | 165.93 |
| + AGENTBALANCE | **74.23** | **82.44** | **85.94** | **88.67** | **1.430** | **64.57** | **82.13** | **86.78** | **88.67** | **173.61** |
| AutoGen (Wu et al., 2024) | 68.91 | 81.70 | 85.19 | 86.78 | 1.449 | 59.87 | 80.74 | 87.04 | 87.37 | 155.13 |
| + AGENTBALANCE | **71.80** | **82.43** | **85.60** | **89.01** | **1.473** | **73.05** | **82.99** | **87.65** | **89.76** | **160.55** |
| MAC-SQL (Wang et al., 2025a) | 37.52 | 44.87 | 46.34 | 47.43 | 0.095 | 40.57 | 43.77 | 44.62 | 53.96 | 10.99 |
| + AGENTBALANCE | **38.11** | **45.78** | **48.84** | **51.89** | **0.099** | **43.13** | **44.87** | **47.95** | **54.13** | **11.86** |




| | $AUC_{tok}$ | $AUC_{lat}$ |
|---|---|---|
| Base Setting | 1.297 | 250.0 |
| Remove LLMs | 1.287 | 247.2 |
| Add New LLMs | 1.318 | 255.2 |

Figure 4: Inductive ability analysis of AGENTBALANCE on MMLU. (a)-(c) show the proportions of LLM invocations under different LLM configurations. $\text{AUC}_{\text{tok}}$ and $\text{AUC}_{\text{lat}}$ are listed on the right.

## 5 Related Work

**LLM-based Multi-Agent Systems.** LLM-based multi-agent systems coordinate role-specialized agents through tool use, message passing, and workflow execution to improve problem solving beyond single-LLM inference (Yao et al., 2023; Schick et al., 2023). Recent MAS design studies mainly target two stages: constructing task-specific agent teams by searching, generating, or initializing roles and templates (Chen et al., 2024; Yuan et al., 2025; Tian et al., 2025), and optimizing execution structures through workflow orchestration (Zhang et al., 2025d; Niu et al., 2025; Shang et al., 2025) or communication-topology search (Zhuge et al., 2024; Zhang et al., 2025a; Yang et al., 2025a; Shen et al., 2025). While effective, these methods mostly assume homogeneous backbones or optimize agent structures independently of backbone heterogeneity, leaving the interaction between heterogeneous LLMs and token-cost & latency budgets underexplored. In contrast, AGENTBALANCE focuses on budget-aware MAS instantiation by jointly assigning heterogeneous backbones, pruning agents, and synthesizing communication topology.

**Cost-effectiveness in MAS.** Cost-effective MAS aims to reduce the overhead from repeated LLM calls, long interaction chains, inter-agent communication, and expensive backbones. Existing methods mainly reduce communication or agent redundancy: AgentPrune (Zhang et al., 2025b) and G-Designer (Zhang et al., 2025c) prune or synthesize communication topologies, while AgentDropout (Wang et al., 2025b) removes less useful agents. Other works introduce backbone selection into MAS, including MasRouter (Yue et al., 2025), BAMAS (Yang et al., 2025b), and DIMA (Ning et al., 2025); however, they either handle limited backbone heterogeneity or lack query-adaptive allocation under varying token-cost and latency budgets. Practical deployments often avoid this complexity by using fixed mid-sized backbones to keep cost and latency bounded (Embar et al., 2025; Ma et al., 2025; Shi et al., 2025), but such standardization cannot adapt model capacity to query difficulty or role-specific demands. In contrast, AGENTBALANCE explicitly optimizes heterogeneous backbone assignment and communication topology under token-cost and latency budgets.

## 6 Conclusion

This paper presents AGENTBALANCE, a budget-aware co-design framework of backbone assignment and communication design for constructing cost-effective MAS under token-cost and latency budgets. AGENTBALANCE first conducts backbone-oriented agent generation to construct candidate agents with heterogeneous LLMs, then performs Adaptive MAS Topology Generation to build cost-effective communication structures. Experiments demonstrate that AGENTBALANCE achieves consistent performance gains across token-cost and latency regimes, while serving as a plug-in to enhance existing MAS and generalizing to unseen LLM backbones, suggesting a practical path toward budget-aware MAS deployment.

## Limitations

AGENTBALANCE focuses on budget-aware instantiation within a predefined MAS design space. It assumes access to agent templates, role instructions, optional tools, and admissible communication orders or edges. Under this setting, AGENTBALANCE can activate templates, assign backbones, and retain communication links, but it does not directly construct a MAS from scratch, such as discovering new roles, designing new tools, or synthesizing entirely new workflows. It also adopts a staged co-design factorization with pool-level backbone selection, feasible masking, hop-length control, and DAG-based topology parameterization for stable optimization. These choices improve tractability and help maintain token-cost and latency feasibility, but may exclude flexible configurations, such as unrestricted cross-pool role assignments, alternative agent orderings, or cyclic refinement-style communication. Relaxing these assumptions while preserving budget controllability and training stability remains future work.

## References


Chris Buckley and Ellen M. Voorhees. 2000. Evaluating evaluation measure stability. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. 2024. Autoagents: A framework for automatic agent generation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Yiqun Chen, Jiaxin Mao, Yi Zhang, Dehong Ma, Long Xia, Jun Fan, Daiting Shi, Zhicong Cheng, Simiu Gu, and Dawei Yin. 2025. Ma4div: Multi-agent reinforcement learning for search result diversification. In *Proceedings of the ACM Web Conference 2025 (WWW '25)*.

Varsha Embar, Ritvik Shrivastava, Vinay Damodaran, Travis Mehlinger, Yu-Chung Hsiao, and Karthik Raghunathan. 2025. LLM-Based Insight Extraction for Contact Center Analytics and Cost-Efficient Deployment. *arXiv preprint arXiv:2503.19090*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. In *International Conference on Learning Representations (ICLR)*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

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. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. 2024. Routerbench: A benchmark for multi-llm routing system. In *Agentic Markets @ ICML 2024 (Workshop)*.

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. 2015. Deep neural decision forests. In *Proceedings of the IEEE International Conference on Computer Vision*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP '23)*.

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

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023b. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems*.

Christos Louizos, Max Welling, and Diederik P. Kingma. 2018. Learning sparse neural networks through L_0 regularization. In *International Conference on Learning Representations (ICLR)*.

Yingwei Ma, Qingping Yang, Rongyu Cao, Binhua Li, Fei Huang, and Yongbin Li. 2025. Alibaba lingmaagent: Improving automated issue resolution via comprehensive repository exploration. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*.

Kaisa Miettinen. 1999. *Nonlinear Multiobjective Optimization*. International Series in Operations Research & Management Science.

Yansong Ning, Shuowei Cai, Wei Li, Jun Fang, Naiqiang Tan, Hua Chai, and Hao Liu. 2025. Dima: An llm-powered ride-hailing assistant at didi. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*.

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. 2025. Flow: Modularized agentic workflow automation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2025. Scaling large language model-based multi-agent collaboration. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Hirad Baradaran Rezaei, Fethi Rabhi, and Amin Beheshti. 2025. Pipemind: Toward a multi-agent framework for real-time feedback and continuous optimization in analytics pipelines. In *Companion Proceedings of the ACM Web Conference 2025*, pages 2891–2894.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. 2025. Agentsquare: Automatic LLM agent search in modular design space. In *The Thirteenth International Conference on Learning Representations (ICLR 2025), Singapore, April 24–28, 2025*.

Xu Shen, Yixin Liu, Yiwei Dai, Yili Wang, Rui Miao, Yue Tan, Shirui Pan, and Xin Wang. 2025. Understanding the information propagation effects of communication topologies in llm-based multi-agent systems. *arXiv preprint arXiv:2505.23352*.

Binpeng Shi, Yu Luo, Jingya Wang, Yongxin Zhao, Shenglin Zhang, Bowen Hao, Chenyu Zhao, Yongqian Sun, Zhi Zhang, Ronghua Sun, Haihua Li, Wei Song, Xiaolong Chen, Jingbo Miao, and Dan Pei. 2025. Flowxpert: Expertizing troubleshooting workflow orchestration with knowledge base and multi-agent coevolution. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2025)*.

SiliconFlow. 2025. SiliconFlow official website. https://siliconflow.cn/. Company website. Accessed: 2025-09-17.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Chunhao Tian, Yutong Wang, Xuebo Liu, Zhexuan Wang, Liang Ding, Miao Zhang, and Min Zhang. 2025. Agentinit: Initializing llm-based multi-agent systems via diversity and expertise orchestration for effective and efficient collaboration. In *Findings of the Association for Computational Linguistics: EMNLP 2025*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025a. Mac-sql: A multi-agent collaborative framework for text-to-sql. In *Proceedings of the 31st International Conference on Computational Linguistics*.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. 2025b. Agent-Dropout: Dynamic agent elimination for token-efficient and high-performance LLM-based multi-agent collaboration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*.

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. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

Jiaxi Yang, Mengqi Zhang, Yiqiao Jin, Hao Chen, Qingsong Wen, Lu Lin, Yi He, Weijie Xu, James Evans, and Jindong Wang. 2025a. Topological structure learning should be a research priority for llm-based multi-agent systems. *arXiv preprint arXiv:2505.22467*.

Liming Yang, Junyu Luo, Xuanzhe Liu, Yiling Lou, and Zhenpeng Chen. 2025b. Bamas: Structuring budget-aware multi-agent systems. *arXiv preprint arXiv:2511.21572*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. 2025. Evoagent: Towards automatic multi-agent generation via evolutionary algorithms. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. 2025. 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)*.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, LEI BAI, and Xiang Wang. 2025a. Multi-agent architecture search via agentic supernet. In *Forty-second International Conference on Machine Learning*.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. 2025b. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. 2025c. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *Forty-second International Conference on Machine Learning*.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, XiongHui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025d. Aflow: Automating agentic workflow generation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Kaiya Zhao, Michelangelo Naim, Jovana Kondic, Manuel Cortes, Jiaxin Ge, Shuying Luo, Guangyu Robert Yang, and Andrew Ahn. 2023. Lyfe agents: Generative agents for low-cost real-time social interactions. *arXiv preprint arXiv:2310.02172*.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. GPTSwarm: Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*.

Table 3: Ablation on AGENTBALANCE. We conduct evaluations with three metrics: Perf (%) (↑), Token-cost (USD) (↓) Latency (s) (↓).

| Method | MATH | | | MMLU | | |
|---|---|---|---|---|---|---|
| | Perf↑ | Tok↓ | Lat↓ | Perf↑ | Tok↓ | Lat↓ |
| AGENTBALANCE | 0.758 | 0.496 | 99.3 | 0.856 | 0.315 | 63.5 |
| Variant (1) | 0.733 | 0.838 | 169.4 | 0.824 | 0.394 | 69.7 |
| Variant (2) | 0.713 | 0.295 | 53.5 | 0.800 | 0.202 | 50.1 |
| Variant (3) | 0.746 | 0.871 | 141.1 | 0.856 | 0.351 | 74.6 |
| Variant (4) | 0.752 | 0.761 | 131.1 | 0.856 | 0.371 | 77.6 |
| Variant (5) | 0.754 | 0.548 | 114.3 | 0.850 | 0.418 | 77.6 |
| Variant (6) | 0.758 | 0.815 | 141.1 | 0.861 | 0.564 | 99.3 |
| Variant (7) | 0.744 | 0.626 | 171.3 | 0.850 | 0.319 | 88.6 |

# A Code availability.

For anonymous review, our open-source code is available at https://anonymous.4open.science/r/AB-UnderReview/.

# B Additional Experiments

## B.1 Framework Analysis

**Ablation Study** We consider seven variants for the ablation study: (1) constructing random LLM pools, (2) random pool selection, (3) random role-backbone matching, (4) enforcing a unified backbone across all agents, (5) using role embeddings only in adaptive MAS topology generation, (6) removing agent gating, and (7) apply dense communication topology. The results are present in Table 3: (1) Constructing random pools or (2) a random pool selection causes a substantial drop in performance. (3) random role-backbon matching,(4)using unified backbone and (5) ignoring backbone/query information in topology generation markedly reduce cost-effectiveness. Furthermore, (6) eliminating agent gating or (7) applying a dense topology leads to significant latency increases.

**Sensitivity Analysis** We present the effect four hyperparameters of AGENTBALANCE in Figure 5(a)–(d), and we highlight the key findings below. **(a) Token-cost weight $\lambda_{\text{tok}}$:** As $\lambda_{\text{tok}}$ increases, decisions shift toward cheaper configurations; correspondingly, token-cost decreases, with latency and accuracy declining slightly (as small backbones are encouraged in cost-effective agent generation). This hyperparameter offers a balanced trade-off to achieve a specific budget. **(b) Latency weight $\lambda_{\text{lat}}$:** With higher $\lambda_{\text{lat}}$, slow configurations are discouraged, which typically lowers latency. In return, accuracy may drop, whereas relaxing this weight yields the opposite behavior. **(c) Difficulty offset $\delta$:** As $\delta$ increases, the selector is more likely

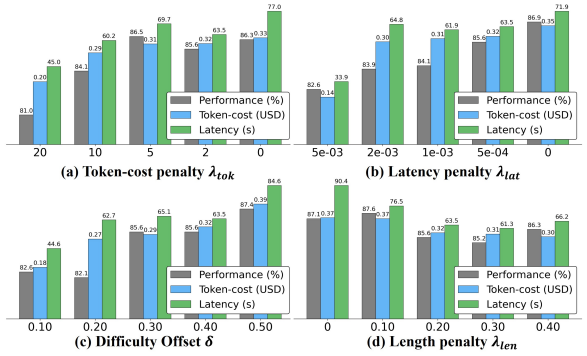



Figure 5: Hyperparameter analysis of AGENTBALANCE.

to choose heavier pools; consequently, accuracy tends to improve mildly, whereas token-cost and latency rise. **(d) Length penalty $\lambda_{\text{len}}$:** By penalizing long communication chains, larger $\lambda_{\text{len}}$ steadily reduces latency.

## B.2 Case Study

We conduct a case study of AGENTBALANCE on three tasks with different resource budgets. As shown in Figure 6, we observe that AGENTBALANCE (i) *performs diverse backbone assignment and topology generation*: as shown in the high budget setting of (a), it places 32B backbones on the other agents while attaching a lighter *Inspector*. It also generates a layered topology in (a) and a parallel topology in (b), respectively, demonstrating its capability to adapt to diverse environments; (ii) *instantiates the MAS according to budget*: as shown in (b), the low-budget configuration keeps a lean two-agent pipeline without an LRM, whereas the high-budget configuration adds verification agents and enables LRMs as backbones; (iii) *exhibits structural stability across budgets*: the core task structures remain stable while capacity increases. *e.g.*, in general QA the three selected agents are retained across budgets, indicating that AGENTBALANCE learns task-level structural information to construct the MAS.

## B.3 Cross-Vendor Robustness

To evaluate AGENTBALANCE in a cross-vendor setting, we construct a multi-vendor candidate pool by aggregating all 23 available LLMs in our setup, spanning heterogeneous capability profiles as well as token-cost/latency regimes. We report results on MMLU and MATH, and compare against the strongest baseline on each dataset. As shown in Table 4, AGENTBALANCE consistently outperforms the strongest baseline under *both* token-cost

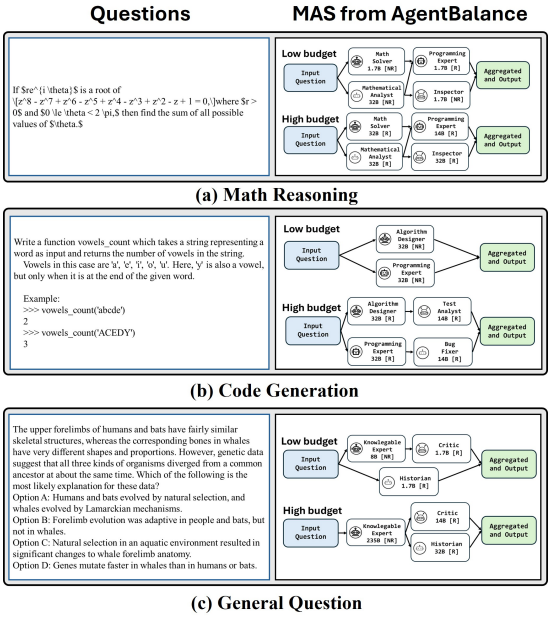



Figure 6: Case study of AGENTBALANCE. Backbones are from the Qwen3 family, with [R] for reasoning models and [NR] for non-reasoning models.

and latency budgets. For example, under tight latency constraints on MMLU, $P@L_1$ improves from 63.26 to 83.18. Moreover, AGENTBALANCE yields higher AUC on both axes (MMLU: $\text{AUC}_{tok}$ $1.291 \rightarrow 1.321$, $\text{AUC}_{lat}$ $249.9 \rightarrow 258.1$; MATH: $\text{AUC}_{tok}$ $3.515 \rightarrow 3.551$, $\text{AUC}_{lat}$ $604.3 \rightarrow 609.3$). These results also support the effectiveness of our backbone modeling is vendor-agnostic, where each candidate is represented by its calibrated $\langle \text{Perf}, \text{TokCost}, \text{Lat} \rangle$ and profile rather than vendor identity, enabling robust optimization over arbitrary mixtures in realistic multi-vendor settings.

### B.4 Scalability and Training Cost

**Scalability.** We further assess scalability by varying the initial template size on MMLU from 5 to $10/15/20$ while keeping the hyperparameter configuration fixed. As shown in Table 5, token-cost and latency grow steadily with the template size, while task performance shows no significant gain. This suggests that, as the template pool grows, the overhead increases in a controlled rather than explosive manner.

**Training cost.** We also report the end-to-end training overhead of AGENTBALANCE. As shown in Table 6, a single setting uses only 5 training iterations. With this lightweight regime, the training overhead remains a controlled one-time cost.

## C Experimental Setup

In this section, we provide more details of experiments setup.

### C.1 Candidate LLMs

We summarize the candidate backbones and their per-token prices (PTP) in Table 7. Most LLM backbones are accessed via online API, while two small models (Qwen3-8B and Qwen3-1.7B) are served locally with `vLLM` (Kwon et al., 2023) using one NVIDIA A100 GPU. For online LLMs, we include models from two vendors: TogetherAI-hosted for GPT-oss-120B, Kimi-K2-Instruct-0905 (T), and Kimi-K2-Thinking-Turbo (T), and Siliconflow (SiliconFlow, 2025) for others to form a multi-vendor candidate backbone pool. For consistent budgeting across providers, we report PTP in USD per million tokens: SiliconFlow prices (quoted in CNY) are converted using USD_PER_CNY$= 0.14024$, while TogetherAI prices are natively in USD. We price the two locally served Qwen3 models using their corresponding SiliconFlow PTP to keep the overall cost consistent.

Our candidate backbones spans 1.7B to 571B parameters and includes both non-reasoning and reasoning types. Beyond the primary LLM backbones for main experiments (Section 4.2), we include additional models for inductive analysis (Section 4.4) and multi-vendor evaluation (Appendix B.3), including Qwen3-Next-80B-A3B, DeepSeek-V3/R1, GLM-4.5 (and GLM-4.5-Air), GLM-4-32B-0414, and Kimi-K2 variants. We also include suboptimal models including Qwen2.5-72B-Instruct for completeness, though it is dominated under our Pareto-based pool construction and thus never selected at runtime. Overall, these candidate backbones enable an evaluation AGENTBALANCE under heterogeneous size, types, vendors and families and while using a unified currency for budget constraints.

### C.2 Details of Implementation

**AgentBalance.** To evaluate performance under different resource budgets, we vary the hyperparameters of AGENTBALANCE as summarized in Table 9. Unless otherwise specified, we set the length penalty to $\lambda_{\text{len}} = 0.2$ and use the configuration of `upper bound`$= 2$. For data efficiency and fair comparison, each method is trained with 20 samples on HUMANEVAL and 40 samples on the other datasets, following the lightweight training

Table 4: AGENTBALANCE with 23 heterogeneous multi-vendor backbones on MMLU and MATH under token-cost and latency budgets. We report performance at $P@T_{1..4}$ and $P@L_{1..4}$ (removing the largest budget point on each axis), along with AUCs. All metrics are higher-is-better (↑); the best per dataset/column is in **bold**.

| | | Under Token-Cost Budgets | | | | | Under Latency Budgets | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Dataset** | **Method** | $P@T_1$ | $P@T_2$ | $P@T_3$ | $P@T_4$ | $\text{AUC}_{tok}$ | $P@L_1$ | $P@L_2$ | $P@L_3$ | $P@L_4$ | $\text{AUC}_{lat}$ |
| MMLU | G-Designer | 57.88 | 68.22 | 82.08 | 88.19 | 1.291 | 63.26 | 78.63 | 84.63 | 88.32 | 249.9 |
| | AGENTBALANCE | **60.80** | **73.83** | **83.02** | **89.10** | **1.321** | **83.18** | **88.89** | **89.31** | **89.73** | **258.1** |
| MATH | AgentPrune | 67.11 | 72.46 | 74.71 | **78.88** | 3.515 | 60.08 | 66.64 | 71.78 | 72.40 | 604.3 |
| | AGENTBALANCE | **70.45** | **73.38** | **75.03** | 78.32 | **3.551** | **64.60** | **75.29** | **78.40** | **78.95** | **609.3** |

Table 5: Scalability of AGENTBALANCE on MMLU under different initial template sizes.

| Template size | 5 | 10 | 15 | 20 |
|---|---|---|---|---|
| Token-cost (USD) | 0.31 | 0.92 | 1.38 | 2.07 |
| Latency (s) | 63.5 | 111.3 | 156.8 | 201.6 |

Table 6: Training overhead of AGENTBALANCE under a single setting.

| Dataset | Cost (USD) | Time (min) | Iterations |
|---|---|---|---|
| MMLU | 0.35 | 125 | 5 |
| MATH | 0.50 | 200 | 5 |
| HumanEval | 0.15 | 30 | 5 |

strategy in (Zhang et al., 2025b; Yue et al., 2025).

**Baselines.** For single-LLM baselines (*i.e.*, AgentPrune, G-designer and AgentDropout), we instantiate the MAS with a single LLM and, for each candidate LLM in turn, evaluate and record the triplets $(\text{Perf}, \text{token-cost}, \text{latency})$. For a chosen budget type (token-cost or latency), we construct the budget–performance frontier by taking the upper envelope (convex hull) of the $(\text{budget}, \text{performance})$ point. This frontier substantially strengthens single-LLM baselines by budget-sweeping over candidate backbones and reporting the upper envelope, *i.e.*, the best achievable performance at each budget level. For MasRouter, we vary $\lambda$ within $\{0, 5, 40, 100\}$ to launch four experiments to match the setting of AGENTBALANCE.

For data efficiency and fair comparison, each method is trained with 20 samples on HumanEval and 40 on the others datasets.

### C.3 Details of Metrics

**Performance-at-Budget.** We present the formulation of Performance-at-Budget (P@·) (Buckley and Voorhees, 2000) here. Let $\mathcal{H} = \{(b_j, p_j)\}_{j=1}^{J}$ denote the frontier points for a given budget type, sorted by budget $b_j$. For any target budget $B$ with $b_\ell \le B \le b_{\ell+1}$, Performance-at-Budget is obtained by linear interpolation between adjacent frontier points:

$$P(B) \;=\; p_\ell \;+\; \frac{B - b_\ell}{b_{\ell+1} - b_\ell}\,\big(p_{\ell+1} - p_\ell\big), \quad (11)$$

Budgets in (P@·) are applied separately for token-cost $P_{\text{T}}(B_{\text{tok}})$ and latency $P_{\text{L}}(B_{\text{lat}})$.

**Area Under Curve (AUC)** We quantify the global cost-effectiveness by integrating the area under each method's performance—budget frontier. As all these methods are sampled at different budget points, we first convexify each method's $(\text{budget}, \text{performance})$ scatter and use the upper envelope (piecewise linear) as its frontier. For comparability, we (i) anchor every frontier at $(0, 0)$; (ii) define a *common integration window* $[0, B_{\max}^{\text{base}}]$, where $B_{\max}^{\text{base}}$ is the base MAS's maximum budget; and (iii) *ensure the right endpoint at $B_{\max}^{base}$ is included in the area*: we append the base MAS's measured point $\big(B_{\max}^{\text{base}}, p_{\max}^{\text{base}}\big)$ to every method's frontier before integration, to control the influence of the high-cost region. We then integrate the area under each truncated frontier over $[0, B_{\max}^{\text{base}}]$, reporting AUC separately for token-cost and for latency (higher is better within the shared budget range).

## D Details of Pool Construction and Profiling

We provide additional details for *Pool Construction with Profiling*, including backbone calibration, Pareto-based filtering, pool clustering, and textual profile encoding.

**Backbone Calibration and Pareto Filtering.** We aggregate candidate LLMs with similar *performance*, *token-cost*, and *latency* into resource pools. For each backbone $b \in \mathcal{B}$, we construct a unified profile

$$\mathbf{z}_b = \langle \text{Perf}_b, \text{TokCost}_b, \text{Lat}_b \rangle.$$

Here, $\text{Perf}_b$ is estimated from benchmark performance on the corresponding task domain,

Table 7: LLM backbones and unit prices used in AGENT-BALANCE. PTPs are in USD per million tokens. 'In' and 'Out' denote input and output PTPs, respectively. 'R' and 'NR' denote reasoning and non-reasoning modes, respectively; 'R/NR' means both modes are available with the same pricing. TogetherAI models are marked with '(T)'. Prices originally quoted in CNY are converted with USD_PER_CNY$= 0.14024$.

| Model | Type | In | Out |
|---|---|---|---|
| **Primary backbones** | | | |
| Qwen3-235B-A22B | R/NR | 0.351 | 1.402 |
| Qwen3-32B | R/NR | 0.140 | 0.561 |
| Qwen3-14B | R/NR | 0.070 | 0.280 |
| Qwen3-8B | R/NR | 0.035 | 0.140 |
| Qwen3-1.7B | R/NR | 0.014 | 0.056 |
| Qwen2.5-72B-Instruct | NR | 0.579 | 0.579 |
| **Inductive analysis** | | | |
| Qwen3-Next-80B-A3B | NR | 0.140 | 0.561 |
| DeepSeek-V3 | NR | 0.140 | 0.561 |
| DeepSeek-R1 | R | 0.280 | 1.122 |
| **Multi-vendor** | | | |
| GLM-4.5 | R/NR | 0.491 | 1.963 |
| GLM-4.5-Air | NR | 0.140 | 0.841 |
| GLM-4-32B-0414 | R | 0.265 | 0.265 |
| Kimi-K2-Thinking | R | 0.561 | 2.244 |
| Kimi-K2-Instruct | NR | 0.561 | 2.244 |
| GPT-oss-120B (T) | NR | 0.150 | 0.600 |
| Kimi-K2-Thinking (T) | R | 1.200 | 4.000 |
| Kimi-K2-Instruct (T) | NR | 1.000 | 3.000 |

$\mathrm{TokCost}_b$ is estimated from its per-token price (PTP) and expected token usage, and $\mathrm{Lat}_b$ is estimated from calibrated inference latency.

Although PTP is usually available from model providers, the actual token-cost and latency during MAS execution depend on completion length, reasoning mode, and serving environment. Therefore, when available, we run a small-sample calibration with 2–4 queries per model to obtain rough estimates of prediction token-cost and inference latency. In addition, we estimate latency from the model's activated parameters and estimate token-cost from expected completion length. For conventional LLMs, we assume comparable completion lengths within the same task domain; for LRMs, we approximate their expected output length as $\gamma_{\text{task}} > 1$ times that of conventional LLMs, following observations from benchmarks of LRMs.

Given $\mathbf{z}_b$ for all candidate backbones, we first remove dominated models using the 3D Pareto frontier over performance, token-cost, and latency. This filtering removes models that are simultaneously weaker and more expensive or slower than another candidate, improving the quality of the subsequent resource pools. For example, *Qwen2.5-72B-Instruct* is discarded in our profiling because it is slower and costlier while underperforming *Qwen3-32B*.

**Pool Clustering.** After Pareto filtering, we cluster the remaining backbones into LLM pools using $k$-medoids over

$$\phi(b) = \left[\mathrm{Perf}_b, \log \mathrm{TokCost}_b, \log \mathrm{Lat}_b\right].$$

The logarithmic transformation is used because token-cost and latency are more meaningful as budget ratios than as absolute differences; for example, we aim to keep the token-cost and latency gap within the same pool within a small multiplicative range such as $5\times$. Each cluster defines one LLM pool, and pools are ordered from weaker/cheaper to stronger/more expensive according to aggregate performance and resource usage.

To avoid highly imbalanced pools, we downsample overpopulated clusters by removing candidates farthest from the medoid and supplement undersized clusters with nearest neighbors when needed. This balancing makes query-conditioned role–backbone matching more stable, because each selected pool contains enough candidates for assigning different backbones to different roles. We report the LLM pools used in all experiments in Table 8. MMLU (inductive1) and MMLU (inductive2) correspond to the settings of removing existing backbones and adding new backbones in the inductive analysis, respectively (Section 4.4); the pools used in the multi-vendor experiments are also included (Appendix B.3).

**Backbone Profiling.** The numerical statistics above are mainly used for Pareto filtering and pool construction. For subsequent MAS construction, we further build textual profiles for each candidate backbone $b$, so that its capability, cost, and reasoning behavior can be encoded into semantic descriptors for matching and optimization. Each backbone is described by three profiles: (i) a *performance profile*, which summarizes benchmark performance and reflects model capability; (ii) a *PTP profile*, which records per-token price and reflects inference cost; and (iii) a *type profile*, which indicates whether the backbone is a conventional LLM or an LRM and captures its reasoning behavior.

The PTP profile is based on the pricing information summarized in Table 7. We encode these three textual profiles with a lightweight TextEncoder, obtaining $e_b^{\text{perf}}, e_b^{\text{ptp}}, e_b^{\text{type}} \in \mathbb{R}^H$. These embeddings provide stable descriptors of backbone capability, cost, and reasoning behavior, and are used by both difficulty-aware pool selection and query-conditioned role–backbone matching.

**Templates of LLM Profiles** We demonstrate the profiles for Qwen3-32B (non-thinking version). We only include the benchmark results that are related to the current task in the performance profile to reduce redundant information. In the PTP profile, we present the per-token price of the LLM. In the type profile, we present the description of the model type. In addition to the conventional LLM profiles fro Qwen3-32B (non-thinking), we also present a LRMs profile in addition:

**Performance Profile**

The Qwen3-32B multilingual large language model (LLM) is a pretrained and instruction tuned generative model with 32 billion parameters.
Its benchmark result for AgentCoding is : BFCL v3 63.00 ; LiveCodeBench v5 31.30 ; CodeForces 1353.

**PTP Profile**

This model has an per-token price of : 1 CNY per million input tokens, 4 CNY per million output tokens.

**Type Profile (For Conventional LLM)**

Standard inference mode with balanced performance, latency, and cost.
No capability of deep reasoning, but fast in response generation and low cost and latency

**Type Profile (For LRM)**

Deep thinking mode that enables more deliberate internal reasoning steps during inference. This mode typically results in longer token usage, significantly increasing both token usage cost and large response latency.

**Algorithm 1:** AGENTBALANCE: High-Level Workflow

**Input :** Dataset $\mathcal{D}$ with queries; base framework $M = (G^*, \mathcal{B})$ with role templates $T$ and candidate backbones $\mathcal{B}$; difficulty estimator $f_{\text{diff}}$; penalties $\lambda_{\text{tok}}, \lambda_{\text{lat}}, \lambda_{\text{len}}$; difficulty offset $\delta$.

**Pool Construction with Profiling (Initialization)**;
$P^* \leftarrow$ CONSTRUCTPOOLS$(\mathcal{B})$;
$P \leftarrow$ BACKBONEPROFILING$(\mathcal{B}, P^*)$;
`// Pareto & clustering; profiling backbones`

**foreach** $Q \in \mathcal{D}$ **do**
  **Pool Selection**;
  $d \leftarrow$ ESTIMATEDIFFICULTY$(f_{\text{diff}}, Q)$;
  $s^\star$, $p_{\text{sel}} \leftarrow$ SELECTPOOL$(P, d, \delta)$;
  `// $s^\star$: pool index`
  **Role-Backbone Matching**;
  $\{m_i\}$, $p_{\text{match}} \leftarrow$ ROLEBACKBONEMATCH$(T, P_{s^\star}, Q)$;
  $V_Q \leftarrow$ INSTANTIATEAGENTS$(T, \{m_i\})$;
  **Agent Representation Learning**;
  $H \leftarrow$ LEARNAGENTREPRESENTATIONS$(V_Q, Q)$;
  **Agent Gating**;
  $G$, $p_{\text{gate}} \leftarrow$ GATEAGENTS$(H)$; `// $G$: agent-wise gating mask`
  $V_Q^{\text{gated}} \leftarrow$ APPLYGATES$(V_Q, G)$;
  `// keep $\geq 2$ agents`
  **Latency-Aware Topology Synthesis**;
  $L_{\max} \leftarrow$ PREDICTHOPBUDGET$(H)$;
  $E$, $p_{\text{topo}} \leftarrow$ SYNTHESIZETOPOLOGY$(V_Q^{\text{gated}}, H)$;
  $E \leftarrow$ PRUNETOLATENCY$(E, L_{\max})$;
  **Execute & Update**;
  $y$, Perf, Tok, Lat $\leftarrow$ EXECUTEMAS$(V_Q^{\text{gated}}, E, Q)$;
  $D \leftarrow \{s^\star, \{m_i\}, G, E\}$; `// decision tuple`
  UPDATE$(\theta;\ D,$ Perf, Tok, Lat$;\ \lambda_{\text{tok}}, \lambda_{\text{lat}}, \lambda_{\text{len}})$;
  ; `//` $p_\theta(D|Q) = p_{\text{sel}}\, p_{\text{match}}\, p_{\text{gate}}\, p_{\text{topo}}$

Table 8: LLM pools used by AGENTBALANCE across datasets. Pools are ordered from weaker (Pool 0) to stronger (Pool 3). Qwen3 models with thinking mode (*i.e.*, reasoning type) are suffixed with '-thinking'.

| **Dataset** | **Pool 0 (weak)** | **Pool 1** | **Pool 2** | **Pool 3 (strong)** |
|---|---|---|---|---|
| HumanEval | Qwen3-1.7B<br>Qwen3-8B<br>Qwen3-14B | Qwen3-14B<br>Qwen3-32B<br>Qwen3-235B-A22B | Qwen3-1.7B-thinking<br>Qwen3-8B-thinking<br>Qwen3-14B-thinking | Qwen3-14B-thinking<br>Qwen3-32B-thinking<br>Qwen3-235B-A22B-thinking |
| Bird | Qwen3-1.7B<br>Qwen3-8B<br>Qwen3-14B | Qwen3-14B<br>Qwen3-32B<br>Qwen3-235B-A22B | Qwen3-1.7B-thinking<br>Qwen3-8B-thinking<br>Qwen3-14B-thinking | Qwen3-14B-thinking<br>Qwen3-32B-thinking<br>Qwen3-235B-A22B-thinking |
| MMLU | Qwen3-1.7B<br>Qwen3-1.7B-thinking<br>Qwen3-8B | Qwen3-14B<br>Qwen3-8B-thinking<br>Qwen3-14B-thinking | Qwen3-235B-A22B<br>Qwen3-14B-thinking<br>Qwen3-32B-thinking | Qwen3-235B-A22B<br>Qwen3-32B-thinking<br>Qwen3-235B-A22B-thinking |
| MATH | Qwen3-1.7B<br>Qwen3-8B<br>Qwen3-1.7B-thinking | Qwen3-14B<br>Qwen3-1.7B-thinking<br>Qwen3-8B-thinking | Qwen3-235B-A22B<br>Qwen3-14B-thinking<br>Qwen3-32B-thinking | Qwen3-235B-A22B<br>Qwen3-32B-thinking<br>Qwen3-235B-A22B-thinking |
| MMLU (inductive1) | Qwen3-1.7B<br>Qwen3-1.7B-thinking<br>Qwen3-8B | Qwen3-1.7B-thinking<br>Qwen3-8B-thinking<br>Qwen3-14B-thinking | Qwen3-8B<br>Qwen3-14B<br>Qwen3-14B-thinking | Qwen3-8B-thinking<br>Qwen3-14B-thinking<br>Qwen3-32B-thinking |
| MMLU (inductive2) | Qwen3-1.7B<br>Qwen3-1.7B-thinking<br>Qwen3-8B | Qwen3-1.7B-thinking<br>Qwen3-8B<br>Qwen3-8B-thinking | Qwen3-14B<br>DeepSeek-V3<br>Qwen3-Next-80B-A3B | Qwen3-14B-thinking<br>DeepSeek-R1<br>Qwen3-235B-A22B-thinking |
| MMLU (multi-vendor) | Qwen3-1.7B<br>Qwen3-1.7B-thinking<br>Qwen3-8B | Qwen3-14B<br>DeepSeek-V3<br>Qwen3-14B-thinking | Kimi-K2-Instruct-0905<br>GLM-4.5<br>Qwen3-Next-80B-A3B | Kimi-K2-Instruct-0905<br>Kimi-K2-Thinking-Turbo<br>DeepSeek-R1 |
| MATH (multi-vendor) | Qwen3-1.7B<br>Qwen3-1.7B-thinking<br>Qwen3-8B | GPT-OSS-120B(T)<br>DeepSeek-V3<br>Qwen3-14B-thinking | Kimi-K2-Instruct-0905<br>GLM-4.5<br>Qwen3-Next-80B-A3B | Kimi-K2-Instruct-0905<br>Kimi-K2-Thinking-Turbo<br>DeepSeek-R1 |

Table 9: Hyperparameters of AGENTBALANCE per dataset

| Dataset | upper bound | $\lambda_{\text{tok}}$ | $\lambda_{\text{lat}}$ | $\delta$ |
|---|---|---|---|---|
| *HumanEval* | 1 | 100 | $1 \times 10^{-2}$ | 0.3 |
| | 2 | 10 | $5 \times 10^{-3}$ | 0.3 |
| | 3 | 1 | $1 \times 10^{-3}$ | 0.4 |
| | 4 | 0 | 0 | 0.6 |
| *Bird* | 1 | 20 | $3 \times 10^{-3}$ | 0.3 |
| | 2 | 5 | $1 \times 10^{-3}$ | 0.3 |
| | 3 | 1 | $2 \times 10^{-4}$ | 0.4 |
| | 4 | 0 | 0 | 0.6 |
| *MMLU* | 1 | 40 | $5 \times 10^{-3}$ | 0.3 |
| | 2 | 10 | $1 \times 10^{-3}$ | 0.3 |
| | 3 | 2 | $5 \times 10^{-4}$ | 0.4 |
| | 4 | 0 | 0 | 0.6 |
| *MATH* | 1 | 20 | $3 \times 10^{-3}$ | 0.3 |
| | 2 | 5 | $1 \times 10^{-3}$ | 0.3 |
| | 3 | 1 | $2 \times 10^{-4}$ | 0.4 |
| | 4 | 0 | 0 | 0.6 |

## E Workflow of AgentBalance

We present the high-level algorithm workflow in Algorithm 1.

## F Details of Pool Selection

**Difficulty Estimator** We derive supervision from RouterBench (Hu et al., 2024), which aggregates eight domain-specific datasets where each example includes correctness outcomes from multiple LLMs. We define an *ease* score $p_i = \frac{1}{M} \sum_{m=1}^{M} \mathbb{I}\{\text{LLM}_m \text{ answers } x_i \text{ correctly}\} \in [0,1]$ for an example $x_i$ evaluated by $M$ LLMs (a lower $p_i$ indicates higher difficulty). Then, the estimator takes the question $x_i$ as input to learn from these targets.

We balance the training split to approximately follow a 1:1:4 ratio across *Agent Coding*, *Math-Text Reasoning*, and *General Tasks*. Our difficulty estimator model is an MPNet (Song et al., 2020) encoder followed by a lightweight MLP head. We fine-tune only the last four layers of MPNet, using a learning rate of $1 \times 10^{-5}$ for the MPNet parameters and $5 \times 10^{-4}$ for the MLP head, with weight decay $1 \times 10^{-3}$. We predict an ease score $s_i \in [0,1]$ for each example $x_i$ and train against soft targets $p_i$. We minimize the average of two equally weighted terms: binary cross-entropy $\mathrm{BCE}(s_i, p_i)$ (with soft targets) and mean-squared error $\mathrm{MSE}(s_i, p_i)$. The resulting model provides an initial difficulty signal, which is subsequently fine-tuned within our difficulty-aware pool selection module to align with cost-effectiveness objectives.

**Smoothed Pool Assignment.** After computing the cost-aware logits $\ell_p$, we mask pools that exceed the user-specified upper bound by setting their logits to $-\infty$. We then obtain pool weights

$$w = \mathrm{softmax}(\mathrm{mask}(\ell)),$$

and construct monotone thresholds over ordered pools by prefix sums:

$$\mathrm{thr}_p = \sum_{i \le p} w_i, \quad p = 1, \ldots, P-1.$$

These thresholds partition the difficulty interval $[0,1]$ into ordered budget tiers. To make the hard bucket assignment smooth, we use sigmoid gates around the thresholds:

$$G_p(d_{\text{eff}}) = \sigma(\kappa(d_{\text{eff}} - \mathrm{thr}_{p-1})) - \sigma(\kappa(d_{\text{eff}} - \mathrm{thr}_p)),$$

where $\kappa$ controls the sharpness and boundary thresholds are set as $\mathrm{thr}_0 = 0$, $\mathrm{thr}_P = 1$. The selected pool $p^\star$ is sampled from the categorical distribution $G(d_{\text{eff}})$, and its probability is $p_{\text{sel}} = G_{p^\star}(d_{\text{eff}})$.

Table 10: Budgets used for the main experiments (Table 1). $P@T_k$ are token-cost budgets and $P@L_k$ are latency budgets.

| | Token-Cost Budgets (USD) | | | | Latency Budgets (s) | | | |
|---|---|---|---|---|---|---|---|---|
| Dataset | $P@T_1$ | $P@T_2$ | $P@T_3$ | $P@T_4$ | $P@L_1$ | $P@L_2$ | $P@L_3$ | $P@L_4$ |
| MMLU | 0.07 | 0.15 | 0.35 | 0.75 | 20.0 | 42.0 | 65.0 | 130.0 |
| HumanEval | 0.03 | 0.10 | 0.20 | 0.60 | 6.0 | 11.0 | 40.0 | 80.0 |
| MATH | 0.08 | 0.17 | 0.50 | 1.00 | 27.0 | 52.0 | 105.0 | 210.0 |

Table 11: Budgets used for the plug-in experiments (Table 2). $P@T_k$ are token-cost budgets and $P@L_k$ are latency budgets.

| | Token-Cost Budgets (USD) | | | | Latency Budgets (s) | | | |
|---|---|---|---|---|---|---|---|---|
| Setting | $P@T_1$ | $P@T_2$ | $P@T_3$ | $P@T_4$ | $P@L_1$ | $P@L_2$ | $P@L_3$ | $P@L_4$ |
| MMLU-L | 0.08 | 0.18 | 0.35 | 0.50 | 20.0 | 40.0 | 75.0 | 100.0 |
| MMLU-AG | 0.06 | 0.12 | 0.30 | 0.60 | 20.0 | 40.0 | 75.0 | 100.0 |
| MAC-SQL | 0.02 | 0.04 | 0.06 | 0.08 | 10.0 | 15.0 | 20.0 | 30.0 |

Table 12: Budgets used for the multi-family/vendor backbone experiments (Table 4). $P@T_k$ are token-cost budgets and $P@L_k$ are latency budgets.

| | Token-Cost Budgets (USD) | | | | Latency Budgets (s) | | | |
|---|---|---|---|---|---|---|---|---|
| Dataset | $P@T_1$ | $P@T_2$ | $P@T_3$ | $P@T_4$ | $P@L_1$ | $P@L_2$ | $P@L_3$ | $P@L_4$ |
| MMLU | 0.05 | 0.10 | 0.20 | 0.40 | 25 | 50 | 75 | 100 |
| MATH | 0.10 | 0.20 | 0.40 | 0.80 | 30 | 60 | 90 | 120 |

## G Training Analysis

In this section, we provide additional analysis of the training process of AgentBalance. This section includes (i) the constraint that stabilize the optimization and (ii) a concise summary of the resulting configuration statistics on MMLU under the operating point used in the main experiments.

**Constraint for optimization.** To keep the optimization within the feasible configuration space and reduce the combinatorial search complexity, we apply the following rules when sampling the decision tuple $\mathcal{D} = \{p, \{m_i\}, \{g_i\}, E_Q\}$: (i) *Upper-triangular DAG parameterization*: we only parameterize edges with $i < j$, enforcing acyclicity by construction. (ii) *Hop-length control*: if the sampled topology exceeds $L_{\max}$, we iteratively remove the lowest-probability edge along the current critical path until the constraint is satisfied, and compute $\mathrm{Pen}_{\text{len}}$. (iii) *Pool feasibility masking*: ineligible pools are masked (logits set to $-\infty$) before sampling. (iv) *Minimum active agents*: if fewer than $K$

Table 13: Training statistics on MMLU (final configurations). P90 denotes the 90th percentile over queries.

| Metric | Mean | Median | P90 | Max |
|---|---|---|---|---|
| Longest path length $\ell(E_Q)$ | 2.22 | 2 | 3 | 4 |
| # Active agents $\sum_i g_i$ | 3.46 | 3 | 4 | 5 |
| Normalized DAG density | 31.26% | 30% | 45% | 50% |

agents are activated, we activate the top-$K$ agents by gating probability. These constraint-handling rules stablize the optimization and substantially shrink the effective action space and filter out infeasible samples, which reduces the variance of the sampled configurations and leads to more stable optimization.

**Configuration statistics after training.** We summarize the statistics of the final instantiated configurations. For each query, we compute the longest path length $\ell(E_Q)$, the number of active agents $\sum_i g_i$, and the normalized DAG density $|E_Q|/(N(N-1)/2)$, where $N = |V_Q^{\text{gated}}|$. When computing $E_Q$, we exclude the input/output links between the MAS and the query/answer interface and only count inter-agent communication edges. Table 13 reports the mean, median, P90 (90th percentile), and maximum over queries; all results are collected across independent runs. Overall, the learned configurations well controlled under the prescribed constraints.

## H Evaluation Budgets

We report the evaluation budgets for the main experiments in Table 10, the plug-in experiments in Table 11, and the multi-vendor experiments in Table 12. Token-cost is computed as the total expenditure for answering questions during evaluation, and latency is the recorded average end-to-end latency per question.

We choose budget points only where the neighborhood is well-populated by operating points from multiple methods. Concretely, for each reported budget value, we require that within a $\pm 5\%$ range around it, there are at least 5 feasible runs (main experiment) or at least 3 feasible runs (plug-in experiment and multi-vendor experiment) across the compared methods. This avoids reporting curves based on isolated configurations and yields more stable performance-at-budget estimates.