METERROUTER: BUDGET-AWARE DYNAMIC ROUTING FOR MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems (MAS) built on large language models (LLMs) outperform single-agent systems, but inter-agent communication incurs substantial token overhead. While assigning a large LLM to every agent results in inefficient resource utilization, leaving the system far from the Pareto frontier. The cost of answering a query is set jointly by how many agents are recruited, the role and LLM assigned to each, and communication topology among them. Most existing routers treat cost only implicitly rather than as an explicit optimization constraint. A fixed accuracy–cost preference also cannot express how much a caller is willing to pay for a particular query. We introduce `MeterRouter`, a budget-aware dynamic router that incrementally synthesizes a collaboration graph by selecting each agent's role, LLM, incoming connections, and termination conditioned on the query and a per-query monetary budget. We initialize the router by imitating successful execution graphs, then jointly refine its decisions with a variant of Group Relative Policy Optimization (GRPO) whose reward favors correct answers and penalizes spending and budget overruns. Across five benchmarks and a three-tier LLM pool, `MeterRouter` is **(1) accurate**, achieving performance comparable to the strongest MAS; **(2) economical**, reaching that accuracy at 82% lower cost than the strongest MAS router and extending the accuracy–cost frontier beyond the ceiling of single-agent systems; and **(3) controllable**, staying within the specified budget on at least 99.9% of queries.

1 INTRODUCTION

Large Language Model (LLM)-based agents have demonstrated remarkable capabilities in tackling complex tasks that extend beyond enclosed reasoning and textual boundaries (Schick et al., 2023), with external capabilities like context-aware memory (Zhang et al., 2024), tool use (Qin et al., 2024). However, single-agent architectures often struggle when executing multi-step reasoning (Wei et al., 2022; Valmeekam et al., 2023), adapting to dynamic environments (Yao et al., 2023), or maintaining coherence over extended horizons (Park et al., 2023; Sun et al., 2026). To address these challenges, Multi-Agent Systems (MAS) have emerged as a dominant paradigm, consistently outperforming single-agent by well-crafted inter-agent communication topologies, harnessing collective intelligence (Chen et al., 2024c; Wang et al., 2025a), specialized role allocation (Hong et al., 2024; Qian et al., 2024), and dynamic error recovery (Shinn et al., 2023; Du et al., 2024; Gou et al., 2024).

With the rapid expansion of a highly accessible and heterogeneous LLM landscape, selecting appropriate LLMs based on their capability and cost has increasingly captured the attention of the research community(Yue et al., 2025; Yang et al., 2026; Cai et al., 2025). Although LLM capabilities generally scale with model size , smaller models reduce computational cost (Feng et al., 2025). Moreover, uniform dependency on a single model introduces critical operational vulnerabilities, leading to service outages or demand spikes (Chen et al., 2024a). LLM-routing is proposed to assign the suitable LLM for each query(Dai et al., 2024; Ding et al., 2024; Ong et al., 2025; Feng et al., 2025; Zhang et al., 2026). Leveraging model diversity by strategically assigning heterogeneous LLMs offer a promising approach for MAS construction.

MAS cooperative performance gains rely on inter-agent information transmission, exchange and assimilation (Zhang et al., 2025a). Such extensive prompt-level communication-spanning inter-agent

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

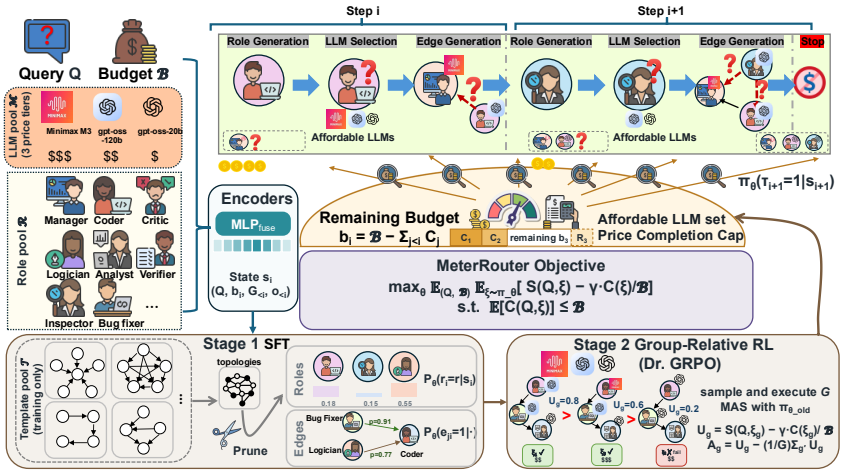

Figure 1: **Overview of `MeterRouter`.** The router builds the MAS incrementally, choosing each agent's role, backbone, and incoming edges, or stopping, under the remaining budget tracked by the meter (top, middle). It is trained by budget-conditioned imitation (Stage 1) and group-relative RL (Stage 2) (bottom).

and self-evaluation introduces substantial token overhead (Wang et al., 2024a),leading to a fundamental trade-off between collaborative synergy and token efficiency. To address that, the MAS topology design has shifted from static, manually designed graphs to more flexible and sparse task-adaptive topology synthesis paradigms(Wang et al., 2025b; Zhang et al., 2025b; Li et al., 2026a). However, these cost-efficiency solutions operate exclusively at the structural orchestration level, uniformly assigning homogeneous LLMs without leveraging heterogeneous LLMs. This motivates a critical question: *How to architect a budget-aware, topologically MAS using heterogeneous backbones that optimizes task performance with cost-efficiency*?

For complex tasks where the division of labor is hard to define, fixed communication topologies including channels cannot meaningfully contribute to collaborative intelligence(Zhang et al., 2025a). In contrast, sparse, task-adaptive graphs reduce redundant roles and edges(Zhang et al., 2025b; Li et al., 2026a). However, designing these topologies independently of model selection can still lead to suboptimal MAS construction. To avoid separating those two selections, we propose `MeterRouter` (Figure 1), a framework that jointly optimizing topologies generation and LLM-to-role assignments. By dynamically tailoring agent topologies (Li et al., 2026a) and LLM assignments by remaining budget, `MeterRouter` optimizes the accuracy-cost tradeoff, achieving competitive task performance while substantially reducing monetary consumption. Our main contributions are as follow: ❶ **Autonomous MAS Orchestration with Heterogeneous LLMs:** We design MAS routing, which dynamic topology generation with assigning each agent's role, backbone, and incoming edges, systematically eliminates the operational redundancies and suboptimal LLM assignments inherent in static, pre-defined templates. ❷ **Budget-Aware and Dynamic adjust:** We introduce a flexible and practical multi-agent framework customized to user-defined budget per query. Instead of cut-off output generation, `MeterRouter` employs a dynamic remaining-budget tracker to adaptively reallocate resources to achieve cost-efficient routing. ❸ **Pareto Optimization:** We introduce `MeterRouter`, a joint routing mechanism that balance task performance and token cost. Driven by a dual-source cost reduction strategy, it minimizes operational overhead by combining structural communication savings with heterogeneous capacity assignment savings.

2 Related Works

Collaboration Topology. MAS have transitioned from fixed interaction heuristics to topology optimization. The progression occurs along two critical axes: **task adaptivity** and **structural flexibility**. Early frameworks relied on static canonical graphs, such as CoT (Wei et al., 2022), Reflexion (Shinn et al., 2023), LLM-debate (Du et al., 2024) and MACNET (Qian et al., 2025). These networks

fundamentally dictate reasoning performance while remaining task-agnostic and computationally inefficient. Dynamic task-adaptive frameworks advance beyond rigid heuristics by leveraging execution feedback to steer communication traces(Hong et al., 2024; Dang et al., 2025; Wu et al., 2024). To eliminate manual topology engineering, template-based optimization frameworks apply learnable edge reweighting and pruning mechanisms to distill redundant communication topologies into sparse structures (Zhuge et al., 2024; Wang et al., 2025b; Zhang et al., 2025a). Pushing beyond template modification, graph generative paradigms construct autonomous topologies: G-Designer (Zhang et al., 2025b) utilizes a graph autoencoder to learn task-adaptive collaboration structures, while ARG-Designer (Li et al., 2026a) leverages an autoregressive model to incrementally build graphs. However, existing MAS frameworks frequently over-provision computational resources by uniformly assigning top-tier LLMs to all agents, leading to excessive token consumption (Yue et al., 2025).

LLM Routing. LLM routing optimizes the Pareto frontier between cost and response quality by assigning the suitable LLMs for queries.Early paradigms rely on sequential cascading, such as FrugalGPT (Chen et al., 2024a), which queries models iteratively until satisfying a confidence threshold, or binary routing based on predicted query difficulty (Ding et al., 2024) and human preference (Ong et al., 2025). Recognizing that ensembling responses across LLMs outperform single LLM, recent works transition from binary model selection to multi-choice routing and output fusion(Dai et al., 2024; Chen et al., 2024b; Feng et al., 2025; Zhang et al., 2026). Extending routing to MAS, MasRouter (Yue et al., 2025) integrates topology selection, role assignment and LLM routing, but it is constrained to selecting from a graph catalog. ProgRouter (Li et al., 2026b) performs step-wise LLM routing across workflow pipelines.

3 Problem Formulation

3.1 Notation Establishment

Search Space.$\mathcal{R}$ denote the pool of available N_r roles (e.g., programmer, analyst), $\mathcal{M}$ denotes N_m available LLM backbone pool, and each LLM backbone carries a price pair $(c_m^{\text{in}}, c_m^{\text{out}})$ in dollars per input and output token. More detailed notation form is in Appendix A

Agents and edges. We represent a multi-agent system as a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The node set $\mathcal{V} = \{v_1, v_2, \cdots, v_N\}$ represents collection of N interacting agents and agent v_i is an instances with an LLM backbone m_i endowed with a specific role function and expertise r_i denoted as $v_i = \{r_i, m_i\}, r_i \in \mathcal{R}, m_i \in \mathcal{M}$. Edges $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$ represent communication pathway. An edge $e_{ji} = (v_j, v_i)$ means that agent v_i is recipient of information from agent v_j. The set of direct predecessors of agent v_i is denoted by $\mathcal{N}_{in}(v_i) = \{v_j | (v_j, v_i) \in \mathcal{E}\}$.

Budget-Aware. For query $\mathcal{Q}$, user has a willingness to pay or budget constraint B. The cost considered in this paper is monetary cost $C_i = c_{m_i}^{\text{in}} T_i^{\text{in}} + c_{m_i}^{\text{out}} T_i^{\text{out}}$. Remaining budget b_i is the money left before agent i, denoted as $b_i = B - \sum_{j<i} C_j$.

Observations. Executing agent i therefore returns an observation. Given the byte counts of information $|\tilde{y}_i|_{\text{B}}$, denoted as $o_i = \Big(|\tilde{y}_i|_{\text{B}}, T_i^{\text{in}}, |y_i|_{\text{chars}}, T_i^{\text{out}} \Big)$, where T_i^{in} is the token count of the message to v_i, $|\tilde{y}_i|_{\text{B}}$ is the size in bytes of the message v_i transmits to its successors, which would cost v_i's successor to read, and agent i outputs message agent i outputs the message y_i, of length $|y_i|_{\text{chars}}$ characters. T_i^{out} is the token count for output. A trajectory ξ records the graph together with everything its execution produced: the responses y_i, the LLM provider-reported token counts T_i^{in} and T_i^{out}, and the per-call costs $C(\xi) = \sum_{i=1}^{N} C_i$.

3.2 Budget-Aware Orchestration

Optimization Goal. Let $\mathcal{P}$ be the distribution over query–budget pairs. The ideal budget-aware MAS should be able to optimize a strategy to maximize utility by considering both performance and marginal cost under the given budget constraints:

$$\begin{aligned} \max_{\theta} \quad & \mathbb{E}_{(\mathcal{Q},B)\sim\mathcal{P}} \mathbb{E}_{\xi\sim\pi_\theta(\cdot|\mathcal{Q},B)} \left[U(\mathcal{Q}, B, \xi) \right] \\ \text{s.t.} \quad & \mathbb{E}_{\xi\sim\pi_\theta(\cdot|\mathcal{Q},B)} \left[C(Q, \xi) \right] \le B, \quad \forall\, (\mathcal{Q}, B) \sim \mathcal{P}, \end{aligned} \tag{1}$$

where π_θ the policy distribution induced by the router and the execution environment. The utility formalizes the trade-off between answer quality and monetary cost. We introduce the utility function: $U(\mathcal{Q}, B, \xi) = S(\mathcal{Q}, \xi) - \gamma \frac{C(\xi)}{B}$, where $S(Q, \xi) \in \{0, 1\}$ indicates whether the answer is correct and γ is tuning parameter reflecting the elasticity of cost sensitivity. The higher value of γ penalizes expenditure more heavily.

MAS Generation In our work, relying on fixed graph templates and deterministic model selection, MeterRouter constructs the MAS as an arbitrary DAG to consider a more general case. To achieve this, `MeterRouter` incrementally expands the expert team by selecting each agent's role, backbone, and incoming edges under the remaining budget progressively. To implement it, we learn a stochastic construction policy π_θ. It induce a distribution over feasible orchestration graphs, which denoted by $P_\theta(\mathcal{G} \mid \mathcal{Q}, B, \mathcal{R}, \mathcal{M}) = P_\theta(\mathcal{G} \mid \mathcal{S})$. We factorize this distribution into a tractable sequence of conditional decisions. At each step, the policy either terminates or adds a new agent with its role, backbone, and incoming edges, conditioned on the current state:

$$P_\theta(\mathcal{G} \mid \mathcal{S}) = \prod_{i \in \mathcal{I}} \underbrace{\pi_\theta(\tau_i \mid s_i)^{I_i^\tau}}_{\text{Termination}} \prod_{i=1}^{|\mathcal{V}|} \left[\underbrace{\pi_\theta(r_i \mid s_i)}_{\text{Role}} \underbrace{\pi_\theta(m_i \mid r_i, s_i)}_{\text{Backbone}} \prod_{j<i} \underbrace{\pi_\theta\left(e_{ji} \mid r_i, m_i, \mathbf{e}_{>j,i}, s_i\right)^{I_{ji}^e}}_{\text{Edge}} \right] \tag{2}$$

Here, $\mathcal{I}$ denotes the reached construction steps. The indicators I_i^τ and I_{ji}^e exclude decisions deterministically imposed by structural or budget constraints from the policy probability. The state before generating the i-th agent, state $s_i = (\mathcal{Q}, B, b_i, \mathcal{G}_{<i}, o_{<i})$, maintains accumulated knowledge and experience from previous interactions by contenting the information of query $\mathcal{Q}$, remaining budget b_i, existing graph $\mathcal{G}_{<i}$, and previous steps observation $o_{<i}$.

There are three types of decisions in the construction: **agent generation**, which predicts the role and selects a backbone from the admissible set $\mathcal{M}_i^m \subseteq \mathcal{M}$ detailed in section 4.2. **edge generation**, which connects the incoming agent with existing agents, and **termination** to stop construction.

4 METERROUTER

Based on the autoregressive graph generation paradigm, the graph generation designed to implement the sequential generation process. `MeterRouter` instantiates π_θ as a small recurrent network (§4.1), making the agent aware available resource limits (§4.2) by tracking the remaining budget It is trained in two stages and run in a generate–execute loop at inference (§4.3).

4.1 MODEL ARCHITECTURE

Input Representation. All textual conditioning information (query, role descriptions and the backbone) is mapped into dense vector representations. The query is mapped into a fixed-dimensional vector $\mathcal{Q} \to \mathbb{R}^D$ by a frozen sentence-BERT text encoder to extract the semantic information, followed by Layer Normalization (LN). f_B denotes the d-dimensional embedding of the log budget, and $\tilde{f}_\mathcal{Q}$ fuses the query with the budget: $f_\mathcal{Q} = \mathrm{MLP}\left(\mathrm{LN}(\mathrm{Enc}(\mathcal{Q}))\right)$, $f_B = \mathrm{MLP}\left(\log B\right)$, $\tilde{f}_\mathcal{Q} = \mathrm{MLP}\left([f_\mathcal{Q}, f_B]\right)$.

Similarly, each available agent role $r_i \in \mathcal{R}$ is converted into $z_r = \mathrm{MLP}\left(\mathrm{LN}(\mathrm{Enc}(r))\right)$, sharing MLP with the query. These are collected into a role embedding matrix $\mathbf{Z} \in \mathbb{R}^{|R| \times d}$. For each LLM $m_i \in \mathcal{M}$, the model information is concatenated with its price, $z_m = \mathrm{MLP}\left([\mathrm{Enc}(m), \log c_m^{\text{in}}, \log c_m^{\text{out}}]\right)$. Encoding the price into the latent vector represents each backbone by what it costs as well as by its description. The edge policy below additionally receives the receiver's input price explicitly.

State Representation. A gated recurrent units (GRU), GRU, aggregates the joint role–backbone encoding of the agents generated so far, $\{[z_{r_j}, z_{m_j}]\}_{j<i}$, forming a historical embedding $f_{\text{hist}}^{(i)}$ in Eq. 10. The meter enters as a cost vector ϕ_i holding the shares of the budget spent and remaining, the shares of the agent cap used and remaining, formed as $\phi_i = \left[\frac{B-b_i}{B}, \frac{b_i}{B}, \frac{i-1}{N_{\max}}, \frac{N_{\max}-i+1}{N_{\max}}, \psi(B)\right]$ with $\psi(B)$ in Eq. 12, the log budget rescaled so that the training range $[B_{\min}, B_{\max}]$ maps to $[-1, 1]$.

$f^{(i)}_{\text{cont}}$ combines the historical information with the current information by composing the budget-conditioned query through a dynamic gate, giving all context information and cost: $f^{(i)}_{\text{cont}} = (1 - g_i) \cdot f^{(i)}_{\text{hist}} + g_i \cdot \tilde{f}_{\mathcal{Q}} + \text{MLP}(\phi_i)$, $g_i = \sigma\left(\frac{f^{(i)}_{\text{hist}} \cdot \tilde{f}_{\mathcal{Q}}}{\sqrt{d}}\right)$, where σ represents the sigmoid function. $f^{(i)}_{\text{edge}}$ in Eq. 11 captures the connectivity pattern of the previous agents by collecting their incoming edges together with a mask of the predecessors already scanned, and encoding them using an MLP.

Another GRU then updates the node-level state $h^{(i)}_{\text{node}}$ based on the context information together with the connectivity of the previous agent, denoted as $h^{(i)}_{\text{node}} = \text{GRU}\Big(\text{MLP}\left([f^{(i)}_{\text{cont}},\, f^{(i)}_{\text{edge}}]\right),\, h^{(i-1)}_{\text{node}}\Big)$.

Role. A metric-learning module projects the hidden state $h^{(i)}_{\text{node}}$ into a "node intent" embedding to predict a node prediction score $s^{(i)}_{\text{node}}$. The $\text{softmax}(\cdot)$ function converts the scores into the probability distribution for role selection:

$$\omega^i_{\text{node}} = \frac{\text{MLP}(h^{(i)}_{\text{node}})^\top \text{MLP}(z_r)}{\sqrt{d}}, \qquad \pi_\theta(r_i = r \mid s_i) = \text{softmax}_{r\in\mathcal{R}}\left(\omega^i_{\text{node}}\right). \tag{3}$$

Backbone. With the role fixed, the router assigns a backbone. Each candidate in the feasible set $\mathcal{M}^m_i$ is scored independently as s^i_m, so no role falls on an unaffordable backbone. $\mathbf{l}_m = (\ell^{\text{in}}_m, \ell^{\text{out}}_m)$ denotes the backbone's log price standardized across $\mathcal{M}$. The pair $\boldsymbol{\psi}(B, b_i) = (\psi(B), \psi(b_i))$ reports the assigned budget and the realized remainder on one log-dollar scale. Because b_i is computed from billed usage, it serves as execution feedback that enters the backbone decision:

$$\omega^i_{\text{m}} = \text{MLP}\left([h^{(i)}_{\text{node}}, z_{r_i}, z_m, f_B, \mathbf{l}_m, \boldsymbol{\psi}(B, b_i)]\right),\ \pi_\theta(m_i \mid r_i, s_i) = \text{softmax}_{m\in\mathcal{M}^m_i}\left(\omega^i_{\text{m}}\right). \tag{4}$$

Edges. With r_i and m_i fixed, the router chooses which existing agents $v_{<i}$ to connect with. A GRU is employed to update the hidden state, initialized from the node-level state $h^{(i,i)}_{\text{edge}} = \text{MLP}\left(\left[h^{(i)}_{\text{node}}, z_{r_i}, z_{m_i}\right]\right)$. For existing agents $\{j = i-1, \ldots, 1\}$, scanned in this reverse order, the edge GRU at each step updates the state based on the previous edge decision and the predecessor's length vector: $h^{(i,j)}_{\text{edge}} = \text{GRU}\Big(\text{MLP}\left(\left[e_{j+1,i}, z_{r_j}, z_{m_j}\right]\right) + \text{MLP}\left(\boldsymbol{\ell}_{j\to i}\right),\, h^{(i,j+1)}_{\text{edge}}\Big)$. where a length vector $\boldsymbol{\ell}_{j\to i}$ built from its observation o_j and the receiver m_i in Eq. 13 to resource usage of this edge. The state $h^{(i,j)}_{\text{edge}}$ then predicts the score for connecting to agent j, which gives the probability of edge e_{ji}:

$$\omega^{i,j}_{\text{edge}} = \text{MLP}(h^{(i,j)}_{\text{edge}}), \qquad \pi_\theta\big(e_{ji} = 1 \mid r_i, m_i, \mathbf{e}_{>j,i}, s_i\big) = \sigma\big(\omega^{i,j}_{\text{edge}}\big). \tag{5}$$

When v_i must close the graph, the edges from the current sinks are fixed to 1 rather than sampled.
Termination. Team size is decided by the node state and the budget, so stopping can respond to a shrinking remainder in the cost vector ϕ_i. The termination policy is evaluated at the start of step i. Termination is forced when the affordable set $\mathcal{M}^m_i = \emptyset$. The probability of termination:

$$\pi_\theta(\tau_i = 1 \mid s_i) = \sigma\big(\text{MLP}([h^{(i)}_{\text{node}}, f_B])\big). \tag{6}$$

4.2 The Meter

Besides tracking the remaining budget in our policy condition, generation is confined to the affordable budget. We therefore form the budget-aware from two sides. One removes unaffordable backbones from the action set, the other converts the remaining money into a hard token cap on the call that is about to be issued. Both run on a byte-based estimator that is deliberately conservative, all constants and estimator forms in this section are given in Appendix B.

Affordable backbone set. Let $\widehat{T}_i$ be an upper estimation of the input node i would consume, if it gets all messages from every completed predecessor. $F_m(T) = c^{\text{in}}_m T + c^{\text{out}}_m q_{\min}$ be the floor cost of a call on m that produces the predefined minimum admissible completion constant $q_{\min}$. Then:

$$\mathcal{M}^m_i = \Big\{m \in \mathcal{M} \ :\ F_m\big(\widehat{T}_i\big) + R_i \le b_i\Big\}, \tag{7}$$

where R_i reserves the floor cost of any successor. If $\mathcal{M}^m_i = \emptyset$, the step terminates. If there is no affordable LLM before the first agent, the query is flagged as budget-blocked without issuing any model call.

Priced completion cap. Once the role, the backbone and the edges are fixed, the input message decided and the remaining budget becomes the hard cap on completion. So the cap decided how long the node i may output in the remaining budget:

$$q_i = \min\Big\{T_{\max},\ \Big\lfloor \frac{b_i - \hat{\rho}_i - R_i}{\mu_i} \Big\rfloor\Big\}, \qquad \mu_i = c^{\text{out}}_{m_i} + \nu_i\, \kappa_{\text{r}}\, c^{\text{in}}_{\underline{m}}, \tag{8}$$

where $T_{\max}$ is the maximum number of completion tokens allowed per call, $\hat{\rho}_i$ is estimated input charge, $\nu_i = 1$ if a successor is still obliged, $\underline{m}$ is the cheapest backbone and κ_{r} converts output tokens into the input tokens they become downstream. The marginal price μ_i can help tighten on exactly the agents whose output will be re-read. Under the estimator, a shielded trajectory satisfies $C(\xi) \le B$ by construction. The estimator is an upper bound.

4.3 Policy Procedure

`MeterRouter` is trained in two separate stages to balance a good initialization against exploration of the full action space. In the **first stage**, the policy learns a prior MAS distribution from a set of pre-generated supervision MAS, $\mathcal{D}_{\text{SFT}}$, by a weighted negative log-likelihood objective function on role and edge decisions in Eq. 14 inspired from Li et al. (2026a) as shown in algorithm 2. This loss has no budget or backbone term, so the budget enters only by the label in $\mathcal{D}_{\text{SFT}}$.

To **build** $\mathcal{D}_{\text{SFT}}$, for every query $\mathcal{Q}$, each template t in a predefined pool $\mathcal{T}$ listed in table 7 is implement K times. Each run draws the roles uniformly and every agent is on the mid-priced backbone. The runs are executed once and re-scored at every budget B of a fixed log-spaced grid by the empirical utility, $\hat{u}_t(\mathcal{Q}, B) = \frac{1}{K}\sum_{k=1}^{K} S(\mathcal{Q}, \xi_{t,k}) - \frac{\gamma}{B}\cdot\frac{1}{K}\sum_{k=1}^{K} C(\xi_{t,k})$. We will always put the templates that maximize $\hat{u}_t(\mathcal{Q}, B)$ and uses cost less than B, to $\mathcal{D}_{\text{SFT}}$, detailed in algorithm 1. Because the label changes with B, $\mathcal{D}_{\text{SFT}}$ pairs every query with its highest-utility affordable template at each budget. The backbone policy is frozen in this stage, so supervision shapes topology and roles, and backbone allocation is left to the second stage.

In the **second stage**, the policy is refined by an on–policy Reinforcement Learning (RL) since its answer can be graded quickly, so that backbone allocation and the response to execution feedback can be learned. For every training query $\mathcal{Q}$, a budget B is drawn log-uniformly from $[B_{\min}, B_{\max}]$ to get a better ability of generalization. The current policy $\pi_{\theta_{\text{old}}}$ then generates and executes a group of G trajectories $\{\xi_i\}_{i=1}^{G}$ under this common budget. Each trajectory is scored by the utility $U_g = U(\mathcal{Q}, B, \xi_g)$, and its advantage in its own group is denoted as $A_g = U_g - \frac{1}{G}\sum_{g'=1}^{G} U_{g'}$.

Let $\mathcal{K}(\xi)$ collect the decisions of ξ made from the policy and $\rho_d(\theta)$ be the ratio of π_θ and $\pi_{\theta_{\text{old}}}$ assigned to decision d. The policy is updated with the clipped surrogate:

$$\mathcal{L}_{\text{RL}} = -\,\mathbb{E}_{\mathcal{Q}, B, \xi_{1:G}\sim\pi_{\theta_{\text{old}}}}\left[\frac{1}{G}\sum_{g=1}^{G}\frac{1}{Z}\sum_{d\in\mathcal{K}(\xi_g)} \min\Big\{\rho_d(\theta)A_g,\ \text{clip}\big(\rho_d(\theta), 1-\epsilon, 1+\epsilon\big)A_g\Big\}\right], \tag{9}$$

where Z is the mean number of decisions in the demonstrations of the first stage, a constant fixed once. The objective function is following Dr. GRPO (Liu et al., 2025) to reward the improvement of utility and isolate the LLM providers' contribution to the likelihood of ξ, which does not depend on θ and totally black box by considering the ratios ρ_d is therefore the exact likelihood ratio of the whole trajectory. All parameters are trained in this stage. The detail can be seen in algorithm 3.

At **inference** the parameters are frozen, and the router receives only the query $\mathcal{Q}$ and the budget B. It builds the multi-agent system online by sampling from π_θ, which makes decision of every step based on the MAS structure and all response before. As Each agent is executed and metered before the next decision is made, every later decision can be made on what the earlier calls actually cost and returned. The answer is the output of the final agent. Detailed information of it is in algorithm 4.

5 Experiments

5.1 Experiments Setup

Datasets. We evaluate `MeterRouter` on three types of tasks: (1) general reasoning, with MMLU-Pro (Wang et al., 2024b), MuSR (Sprague et al., 2024), and BBEH (Kazemi et al., 2025); (2)

mathematical reasoning, with Omni-MATH-2 (Ballon et al., 2026); and (3) code generation, with MBPP+ (Liu et al., 2023). For each benchmark, we split the queries into disjoint training, validation, and test sets (Table 5). Omni-MATH-2, MMLU-Pro, and BBEH each have 1,000 test queries, and MuSR and MBPP+ use every query left after the training and validation splits as their test sets (516 and 138).

Baselines. We compare `MeterRouter` with four groups of methods: (1) single-agent methods, namely vanilla and Chain-of-Thought prompting (Wei et al., 2022); (2) fixed multi-agent topologies, including Chain, Tree, and Complete Graph (Qian et al., 2025), and LLM-Debate (Du et al., 2024); (3) dynamic MAS, including AutoGen (Wu et al., 2024), GPTSwarm (Zhuge et al., 2024), and AgentPrune (Zhang et al., 2025a); and (4) LLM routing methods, including GraphRouter (Feng et al., 2025), RouterDC (Chen et al., 2024b), MasRouter (Yue et al., 2025), and the BAMAS explicitly including budget (Yang et al., 2026).

Backbones LLMs. We build a three-tier LLM pool from gpt-oss-20b, gpt-oss-120b, and MiniMax-M3 (Table 6). Their sizes grow from 20B total parameters for gpt-oss-20b to ∼428B for MiniMax-M3. The two gpt-oss models support 128K-token contexts, while MiniMax-M3 supports up to 1M tokens. Per-token prices rise with tier: from gpt-oss-20b to MiniMax-M3, the input price increases from \$0.06 to \$0.30 and the output price from \$0.19 to \$1.20 per million tokens.

Implementation. The text encoder is all-MiniLM-L6-v2 Wang et al. (2020) and each system has at most $N_{\max} = 5$ agents. Role sets follow the style of ARG-Designer Li et al. (2026a). All LLMs are called with temperature 0.2 and at most $T_{\max} = 32{,}768$ completion tokens per call. We evaluate at $B \in \{1, 5, 50\} \times 10^{-3}$ and add $B = 4 \times 10^{-3}$ for Omni-MATH-2 and BBEH, where accuracy changes most sharply between 1 and 5×10^{-3}. Each benchmark's checkpoint is selected on the validation set, and we report results on the test sets. We access all LLMs through Ollama Cloud API.

5.2 Performance Analysis

Accuracy. Table 1 reports test accuracy and cost per query, with `MeterRouter` at $B = 5 \times 10^{-2}$ and $N_{\max} = 5$. `MeterRouter` reaches an average accuracy of 66.72%, less than one point below the two most accurate baselines, MasRouter (66.80%) and LLM-Debate with gpt-oss-120b (66.70%). It is the most accurate method on MuSR (72.2%) and MBPP+ (86.6%). BBEH is its weakest benchmark: there, LLM-Debate with gpt-oss-120b is 6.3 points more accurate.

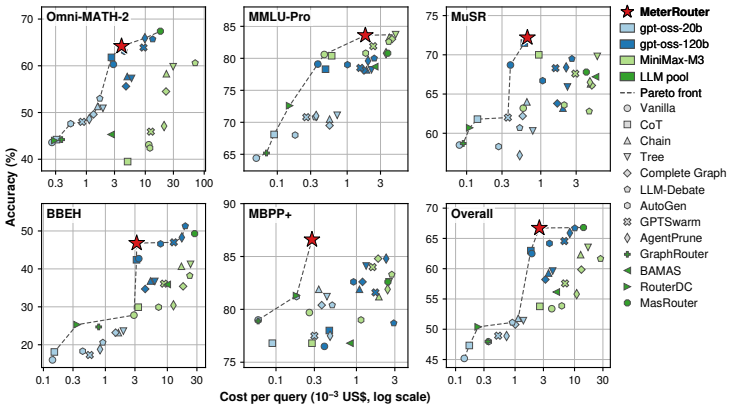

Figure 2: Accuracy versus cost per query on each benchmark and on the overall average. The dashed line is the Pareto frontier; marker shape denotes the method and color the backbone LLM.

Cost–accuracy trade-off. `MeterRouter` spends 2.56×10^{-3} \$ per query on average, $4\times$ less than LLM-Debate and $5.5\times$ less than MasRouter. Fig. 2 shows that it lies on the Pareto frontier of all five benchmarks and of the overall average, so no baseline is both more accurate and cheaper. On the overall front, `MeterRouter` sits at the point where extra spending stops paying off: LLM-Debate adds only 0.47 points for its higher cost. Compared with BAMAS, whose predefined tier-based

Table 1: Comparison with single-agent, multi-agent, and routing baselines. **Multi.**: builds a multi-agent topology; **Router**: selects LLMs from the pool. Acc. is averaged over all test questions, and Cost is the average cost per query. The top five performing methods are highlighted in bold.

Method	Multi.	Router	LLM	Omni-Math-2	MMLU-Pro	MuSR	BBEH	MBPP+	Acc.	Cost ($ $\times 10^{-3}$)
Vanilla	✗	✗	MiniMax-M3	43.1	80.6	63.2	27.8	79.7	53.39	4.14
			gpt-oss-120b	60.3	79.1	68.7	42.7	76.5	62.50	1.91
			gpt-oss-20b	43.6	64.4	58.5	16.0	79.0	45.18	0.14
CoT	✗	✗	MiniMax-M3	39.5	80.4	**70.0**	29.9	76.8	53.78	2.61
			gpt-oss-120b	61.8	78.3	**71.5**	42.4	78.0	62.98	1.84
			gpt-oss-20b	44.2	68.1	61.8	18.1	76.8	47.32	0.17
Chain	✓	✗	MiniMax-M3	58.3	**83.3**	66.3	40.7	81.2	62.32	12.68
			gpt-oss-120b	57.7	78.2	63.2	36.8	81.9	59.28	3.65
			gpt-oss-20b	51.3	70.5	64.0	23.2	81.9	51.81	1.13
Tree	✓	✗	MiniMax-M3	59.8	**83.7**	**69.8**	41.2	82.6	63.52	16.87
			gpt-oss-120b	57.3	78.2	65.9	36.7	**84.1**	59.61	4.30
			gpt-oss-20b	50.9	71.1	60.3	23.6	81.2	51.42	1.38
Complete Graph	✓	✗	MiniMax-M3	54.5	**83.0**	66.1	35.4	**84.8**	59.85	13.03
			gpt-oss-120b	55.6	78.1	63.8	34.7	82.6	58.21	3.24
			gpt-oss-20b	49.6	69.5	62.2	23.2	80.4	50.77	1.01
LLM-Debate	✓	✗	MiniMax-M3	60.6	**82.6**	62.8	38.2	83.3	61.65	27.34
			gpt-oss-120b	**65.7**	80.0	**69.5**	**51.3**	78.7	66.70	10.13
			gpt-oss-20b	53.0	70.8	60.7	20.6	80.4	51.12	0.90
AutoGen	✓	✗	MiniMax-M3	42.4	80.8	63.6	29.9	79.0	53.86	6.10
			gpt-oss-120b	63.1	79.0	66.7	46.6	82.6	64.18	3.79
			gpt-oss-20b	47.6	68.0	58.3	18.3	81.2	47.95	0.36
GPTSwarm	✓	✗	MiniMax-M3	45.9	81.9	67.6	36.1	**84.0**	57.57	6.93
			gpt-oss-120b	**63.9**	78.5	68.3	**47.0**	81.6	64.56	6.74
			gpt-oss-20b	48.0	70.8	62.0	17.3	77.5	48.93	0.52
AgentPrune	✓	✗	MiniMax-M3	47.1	80.8	66.5	30.4	81.9	55.80	10.87
			gpt-oss-120b	**65.9**	79.6	68.4	**48.3**	**84.8**	65.90	8.33
			gpt-oss-20b	48.6	71.0	57.2	18.8	77.5	48.88	0.72
GraphRouter	✗	✓	LLM Pool	44.2	65.2	58.7	24.7	78.9	47.97	0.35
RouterDC	✗	✓	LLM Pool	44.0	72.6	60.7	25.3	81.3	50.39	0.24
BAMAS	✓	✓	LLM Pool	45.3	78.7	67.2	35.9	76.8	56.15	4.93
MasRouter	✓	✓	LLM Pool	**67.4**	80.8	67.8	**49.3**	82.6	66.80	14.15
MeterRouter	✓	✓	LLM Pool	**64.2**	**83.6**	**72.2**	**46.8**	**86.6**	66.72	2.56

preference systematically favors high-tier models (MiniMax-M3), our method can instead select mid- or low-tier models when they perform better.

5.3 Ablation Study

Table 2 varies one factor at a time: whether RL is used, the budget, and the maximum number of agents $N_{\max}$. SFT supplies a prior over topologies and roles, whereas RL learns backbone allocation and how to react to the realized cost. At $N_{\max} = 5$, removing the second-stage RL (w/o RL) lowers the average accuracy from 66.72% to 57.34% and reduces accuracy on all five benchmarks, while the cost stays essentially the same (2.56 vs. 2.55×10^{-3} \$ per query). RL helps just as much at the other two caps, adding 10.16 points at $N_{\max} = 4$ and 8.78 points at $N_{\max} = 6$.

Lowering the budget reduces accuracy monotonically, from 66.72% at $B = 5 \times 10^{-2}$ to 61.42% at 5×10^{-3} and 56.06% at $B \in \{1, 4\} \times 10^{-3}$, and cost falls with it, from 2.56 to 1.96 and 1.53×10^{-3} \$ per query. The decline holds on every benchmark and for every $N_{\max}$. BBEH is the most sensitive to the budget, while MBPP+ is the least. Accuracy drops as the budget shrinks, but nearly every query stays within its budget: across the three agent caps, the average IBR never falls below 99.98%.

The agent cap has little impact on accuracy because it is rarely reached. As shown in Fig. 3 for MMLU-Pro, most runs terminate through the learned stopping policy before hitting the maximum number of agents. Across $N_{\max} \in \{4, 5, 6\}$ the average stays between 65.85% and 66.72%, with $N_{\max} = 5$ the highest. Raising the cap to six adds 0.88×10^{-3} \$ per query (34%) with little change in accuracy. Lowering it to four saves 0.09×10^{-3} \$ per query and lowers accuracy by 0.15 points.

Table 2: Ablation studies of `MeterRouter`. Cost is measured in units of \$ $\times 10^{-3}$.

Method	OmniMath-2	MMLU-Pro	MuSR	BBEH	MBPP+	Acc.	Cost
MeterRouter	64.2	83.6	72.2	46.8	86.6	66.72	2.56
w/o RL	53.7	75.9	67.2	34.2	79.2	57.34	2.55
$B = \$5 \times 10^{-3}$	60.4	81.4	69.4	35.0	85.5	61.42	1.96
$B \in \{\$1, \$4\} \times 10^{-3}$	56.2	73.8	62.4	31.3	82.2	56.06	1.53
Max agents = 4	65.5	82.2	71.6	45.2	84.8	66.08	2.47
w/o RL	52.1	74.5	64.0	33.5	81.2	55.92	2.84
$B = \$5 \times 10^{-3}$	58.0	78.0	68.8	33.7	83.3	59.30	1.47
$B \in \{\$1, \$4\} \times 10^{-3}$	56.3	73.4	58.7	32.5	80.8	55.73	1.27
Max agents = 6	65.1	83.1	69.1	45.1	84.2	65.85	3.44
w/o RL	53.8	75.0	65.7	34.7	79.9	57.07	3.19
$B = \$5 \times 10^{-3}$	57.4	79.9	66.3	35.8	83.2	59.90	1.81
$B \in \{\$1, \$4\} \times 10^{-3}$	54.3	76.3	63.6	31.6	82.1	56.49	1.64

Budget allocation. Fig. 3 Examines how `MeterRouter` allocates additional budget on Omni-MATH-2 and MMLU-Pro when $N_{\max} = 4$, the two benchmarks whose generated systems change most with the budget. The two benchmarks spend the extra budget in different ways. On Omni-MATH-2, the router keeps gpt-oss-120b for about 92% of agents at every budget but builds larger systems: the mean number of agents rises from 1.22 at $B = 5 \times 10^{-3}$ to 3.13 at 5×10^{-2}, and 42.5% of queries use four agents. On MMLU-Pro, the team size stays near 1.4 agents, and the budget instead buys a stronger backbone: the share of agents on MiniMax-M3 rises from 59% at $B = 10^{-3}$ to 93% at 5×10^{-3}. The right panel shows that per-query cost stays within the budget at every level, but the shape of its distribution changes with the budget. At the tight budgets, the distributions are flat at the top because many queries spend close to B but none exceed it. At $B = 5 \times 10^{-2}$, no longer limited by the budget, spending varies by query instead of filling the budget.

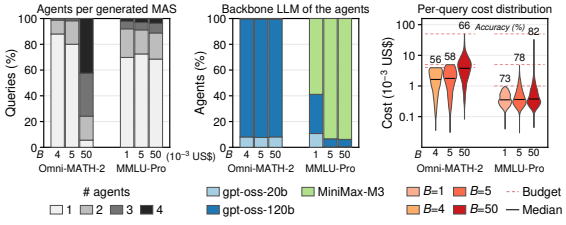

Figure 3: How `MeterRouter` adapts the MAS it generates to the per-query budget B on OmniMath-2 and MMLU-Pro. **Left**: share of queries by the number of agents used. **Middle**: share of agents assigned to each backbone LLM. **Right**: distribution of per-query cost on a log scale. Each violin spans the min to max cost.

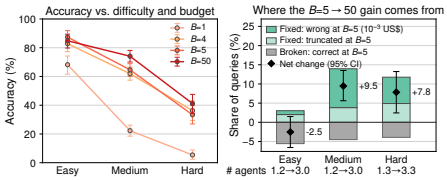

Figure 4: Budget use across problem difficulty on OmniMath-2 (Easy < 4, Medium 4–5.5, Hard $\geq$ 5.5). **Left**: accuracy of `MeterRouter` at each budget. **Right**: change from $B = 5$ to $B = 50$.

Gains by difficulty. Omni-MATH-2 assigns each query a difficulty level, and Fig. 4 breaks down `MeterRouter`'s accuracy by difficulty and budget. The left panel shows that accuracy falls with difficulty at every budget and that the budget matters more on harder problems. The right panel shows that the larger budget, together with larger agent teams, improves accuracy on medium and hard problems but gives little benefit on easy ones.

6 Conclusion

We presented `MeterRouter`, which builds a MAS for each query under a monetary budget, choosing the agents' roles, backbone LLMs, and connections one agent at a time. Because the router observes the realized cost of every call, it makes agents budget-aware to better allocate the remaining resources and removes backbones it can no longer afford, which keeps nearly all queries within budget. Across five benchmarks, `MeterRouter` matches or exceeds existing fixed and dynamic MAS in accuracy at a fraction of their cost.

AI USE STATEMENT

LLMs play two distinct roles in this work. As objects of study: gpt-oss-20b, gpt-oss-120b, and MiniMax-M3 serve as agent backbones in MeterRouter and in all baselines. They are accessed through the Ollama Cloud API with the settings in Section 5.1, and all reported accuracies and costs are produced by these calls. As writing assistants: ChatGPT and Claude were used to polish the grammar and wording of author-written text.

ETHICS STATEMENT

This work aims to improve the cost efficiency and accessibility of multi-agent language model systems through budget-aware routing. Our experiments use publicly available benchmarks and focus on routing among frozen language models. We do not anticipate specific ethical concerns arising from the proposed routing method. However, MeterRouter may inherit biases, factual errors, and harmful outputs from its underlying models. Responsible deployment should therefore include appropriate safeguards, privacy protections for queries sent to external providers, and human oversight in sensitive applications

REPRODUCIBILITY STATEMENT

We have made efforts to ensure the reproducibility of our results. The full method, including the router architecture, the budget meter, and both training stages, is specified in Section 4, with supplementary functions in Appendix B and pseudocode in Appendix C. All five benchmarks are publicly available; data splits are given in Table 5 and the complete role prompts in Appendix F. All backbone LLMs are accessed through Ollama Cloud API, and costs are computed at the fixed per-token rates in Table 6. Hyperparameters of MeterRouter are listed in Table 4, and the configurations of all baselines in Appendix G. Anonymized source code is included in the supplementary material.

REFERENCES

Artificial Analysis. Artificial analysis: Models. https://artificialanalysis.ai/models/, 2026. Accessed: 2026-07-17.

Marthe Ballon, Andres Algaba, Brecht Verbeken, and Vincent Ginis. Benchmarks saturate when the model gets smarter than the judge. CoRR, abs/2601.19532, 2026. URL https://arxiv.org/abs/2601.19532.

Shuowei Cai, Yansong Ning, and Hao Liu. AgentBalance: Backbone-then-topology design for cost-effective multi-agent systems under budget constraints. CoRR, abs/2512.11426, 2025. URL https://arxiv.org/abs/2512.11426.

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, 2024, 2024a. URL https://openreview.net/forum?id=cSimKw5p6R.

Shuhao Chen, Weisen Jiang, Baijiong Lin, James T. Kwok, and Yu Zhang. RouterDC: Query-based router by dual contrastive learning for assembling large language models. In Advances in Neural Information Processing Systems 37 (NeurIPS 2024), 2024b. URL http://papers.nips.cc/paper_files/paper/2024/hash/7a641b8ec86162fc875fb9f6456a542f-Abstract-Conference.html.

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 (ICLR). OpenReview.net, 2024c. URL https://openreview.net/forum?id=EHg5GDnyq1.

Xiangxiang Dai, Jin Li, Xutong Liu, Anqi Yu, and John C. S. Lui. Cost-effective online multi-LLM selection with versatile reward models. CoRR, abs/2405.16587, 2024. URL https://arxiv.org/abs/2405.16587.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration. In Advances in Neural Information Processing Systems 38 (NeurIPS 2025), 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/f1320d2e2842169c6fc89dcbd80e94d0-Abstract-Conference.html.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In International Conference on Learning Representations (ICLR). OpenReview.net, 2024. URL https://openreview.net/forum?id=02f3mUtqnM.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In Proceedings of the 41st International Conference on Machine Learning (ICML), volume 235 of Proceedings of Machine Learning Research, pp. 11733–11763. PMLR, 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Tao Feng, Yanzhen Shen, and Jiaxuan You. GraphRouter: A graph-based router for LLM selections. In International Conference on Learning Representations (ICLR). OpenReview.net, 2025. URL https://openreview.net/forum?id=eU39PDsZtT.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In International Conference on Learning Representations (ICLR). OpenReview.net, 2024. URL https://openreview.net/forum?id=Sx038qxjek.

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

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K. Jain, Virginia Aglietti, Disha Jindal, Peter Chen, Nishanth Dikkala, Gladys Tyen, Xin Liu, Uri Shalit, Silvia Chiappa, Kate Olszewska, Yi Tay, Vinh Q. Tran, Quoc V. Le, and Orhan Firat. BIG-Bench extra hard. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 26473–26501. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1285. URL https://aclanthology.org/2025.acl-long.1285/.

Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 40, pp. 23142–23150, 2026a. doi: 10.1609/aaai.v40i28.39481.

Songyuan Li, Ahmed M. Abdelmoniem, and Shiqiang Wang. ProgRouter: Online progress-guided orchestration for multi-agent LLM workflows under quality-cost tradeoffs. In Findings of the Association for Computational Linguistics: EMNLP 2026. Association for Computational Linguistics, 2026b. URL https://arxiv.org/abs/2608.25992.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In Advances in Neural Information Processing Systems 36 (NeurIPS 2023), 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective, 2025. URL https://arxiv.org/abs/2503.20783.

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

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST), pp. 2:1–2:22. ACM, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 15174–15186. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

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. Scaling large language model-based multi-agent collaboration. In International Conference on Learning Representations (ICLR). OpenReview.net, 2025. URL https://openreview.net/forum?id=K3n5jPkrU6.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs.

In International Conference on Learning Representations (ICLR). OpenReview.net, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Advances in Neural Information Processing Systems 36 (NeurIPS 2023), 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In Advances in Neural Information Processing Systems 36 (NeurIPS 2023), 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Zayne Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. MuSR: Testing the limits of chain-of-thought with multistep soft reasoning. In International Conference on Learning Representations (ICLR). OpenReview.net, 2024. URL https://openreview.net/forum?id=jenyYQzue1.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon agent via context folding. In Proceedings of the 43rd International Conference on Machine Learning. PMLR, 2026. URL https://openreview.net/forum?id=lNRgWoGfYg.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models - A critical investigation. In Advances in Neural Information Processing Systems 36 (NeurIPS 2023), 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/efb2072a358cefb75886a315a6fcf880-Abstract-Conference.html.

Junlin Wang, Siddhartha Jain, Dejiao Zhang, Baishakhi Ray, Varun Kumar, and Ben Athiwaratkun. Reasoning in token economies: Budget-aware evaluation of LLM reasoning strategies. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 19916–19939. Association for Computational Linguistics, 2024a. doi: 10.18653/v1/2024.emnlp-main.1112. URL https://aclanthology.org/2024.emnlp-main.1112/.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In International Conference on Learning Representations (ICLR). OpenReview.net, 2025a. URL https://openreview.net/forum?id=h0ZfDIrj7T.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. Advances in neural information processing systems, 33:5776–5788, 2020.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track, 2024b. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract-Datasets_and_Benchmarks_Track.html.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. AgentDropout: Dynamic agent elimination for token-efficient and high-performance LLM-based multi-agent collaboration. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 24013–24035. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.acl-long.1170. URL https://aclanthology.org/2025.acl-long.1170/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Advances in Neural Information Processing Systems 35 (NeurIPS 2022), 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

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 First Conference on Language Modeling, COLM 2024, 2024. URL https://openreview.net/forum?id=uAjxFFing2.

Liming Yang, Junyu Luo, Xuanzhe Liu, Yiling Lou, and Zhenpeng Chen. BAMAS: Structuring budget-aware multi-agent systems. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 40, pp. 29802–29810, 2026. doi: 10.1609/aaai.v40i35.40226.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In International Conference on Learning Representations (ICLR). OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. MasRouter: Learning to route LLMs for multi-agent systems. In Proceedings of the 63rd Annual Meeting

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

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

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-Designer: Architecting multi-agent communication topologies via graph neural networks. In Proceedings of the 42nd International Conference on Machine Learning, volume 267 of Proceedings of Machine Learning Research, pp. 76678–76692. PMLR, 2025b. URL https://proceedings.mlr.press/v267/zhang25cu.html.

Xuechen Zhang, Zijian Huang, Ege Onur Taga, Carlee Joe-Wong, Samet Oymak, and Jiasi Chen. Efficient contextual LLM cascades through budget-constrained policy learning. In Advances in Neural Information Processing Systems 37 (NeurIPS 2024), 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/a6deba3b2408af45b3f9994c2152b862-Abstract-Conference.html.

Zheyuan Zhang, Kaiwen Shi, Zhengqing Yuan, Zehong Wang, Tianyi Ma, Keerthiram Murugesan, Vincent Galassi, Chuxu Zhang, and Yanfang Ye. AgentRouter: A knowledge-graph-guided LLM router for collaborative multi-agent question answering. In Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 788–809. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.33. URL https://aclanthology.org/2026.acl-long.33/.

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 NOTATIONS

We conclude the used notations in table 3

Table 3: Summary of notation.

Symbol	Description
Input and trajectory	
$\mathcal{Q}$	User query.
B	Cost budget.
$\mathcal{P}$	Distribution of query–budget pairs.
ξ	Multi-agent execution trajectory.
N	Number of agents.
$[B_{\min}, B_{\max}]$	Predefined training budget range.
y_i	Output message of agent i.
$\tilde{y}_i$	Message agent i transmits to its successors.
$T_i^{\text{in}},\ T_i^{\text{out}}$	Input and output token counts of agent i.
$\lvert\cdot\rvert_{\text{B}},\ \lvert\cdot\rvert_{\text{chars}}$	Size in bytes; length in characters.
o_i	Observation of agent i, $\big(\lvert\tilde{y}_i\rvert_{\text{B}}, T_i^{\text{in}}, \lvert y_i\rvert_{\text{chars}}, T_i^{\text{out}}\big)$.
$o_{<i}$	Observations of agents $1, \ldots, i-1$.
Search space and graph	
$\mathcal{R},\ N_r$	Pool of candidate roles and its size.
$\mathcal{M},\ N_m$	Pool of backbone LLMs and its size.
r_i	Agent role.
m_i	Backbone language model.
$c_m^{\text{in}},\ c_m^{\text{out}}$	Price (in USD$) per input and output token of backbone m.
$\underline{m},\ \bar{m}$	Cheapest backbone; mid-priced backbone.
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Multi-agent system as a DAG.
$v_i = \{r_i, m_i\}$	Agent i.
e_{ji}	Edge from agent j to agent i.
$\mathbf{e}_{>j,i}$	Edge decisions of agent i already made in the reverse scan.
$\mathcal{S}$	Space of agent states.
$\mathcal{N}_{in}(v_i)$	Direct predecessors of agent i.
$\mathcal{G}_{<i}$	Partial graph of the first $i-1$ agents.
$N_{\max}$	Maximum number of agents.
$\mathcal{T},\ t$	Pool of topology templates; a template.
Policy decisions	
b_i	Remaining budget.
$s_i = (\mathcal{Q}, B, b_i, \mathcal{G}_{<i}, o_{<i})$	Policy state before step i.
τ_i	Termination decision.
π_θ	Routing policy parameterized by θ.
$P_\theta(\mathcal{G} \mid \cdot)$	Policy-induced distribution over graphs.
$\mathcal{I}$	Construction steps actually reached.
$I_i^\tau,\ I_{ji}^e,\ I_i^r$	Indicators of termination, edge and role decisions sampled from π_θ.
$\mathcal{M}_i^m$	Affordable backbone set at step i.
Optimization	
$S(\mathcal{Q}, \xi)$	Task performance (reward).
C_i	Cost of agent i.
$C(\xi)$	Total trajectory cost.
$U(\mathcal{Q}, B, \xi)$	Utility, $S(\mathcal{Q}, \xi) - \gamma\, C(\xi)/B$.
γ	Cost-sensitivity coefficient.
Model architecture	
$\text{Enc}(\cdot),\ \text{LN}(\cdot)$	Frozen sentence encoder; layer normalization.
$D,\ d$	Encoder output dimension; hidden dimension.
$f_{\mathcal{Q}},\ f_B$	Query embedding; log-budget embedding.

Continued on next page

Symbol	Description
$\tilde{f}_{\mathcal{Q}}$	Budget-conditioned query embedding.
$z_r,\ \mathbf{Z}$	Role embedding; role embedding matrix.
z_m	Backbone embedding.
$\mathbf{l}_m$	Standardized log prices $(\ell_m^{\text{in}}, \ell_m^{\text{out}})$ of backbone m.
$f_{\text{hist}}^{(i)}$	History embedding of generated agents.
ϕ_i	Cost vector.
$\psi(\cdot),\ \boldsymbol{\psi}(B, b_i)$	Rescaled log-dollar map; the pair $(\psi(B), \psi(b_i))$.
$f_{\text{cont}}^{(i)},\ g_i$	Context embedding; its gate.
$f_{\text{edge}}^{(i)}$	Connectivity embedding of previous agents.
$h_{\text{node}}^{(i)}$	Node-level hidden state.
$h_{\text{edge}}^{(i,j)}$	Edge-level hidden state when scanning agent j in agent selecting state i.
$\boldsymbol{\ell}_{j\to i}$	Length vector of edge (v_j, v_i).
$\omega_{\text{node}}^{i},\ \omega_{\text{m}}^{i},\ \omega_{\text{edge}}^{i,j}$	Role, backbone and edge score.
$\text{GRU}_{(\cdot)}$	Gated Recurrent Unit.
$\text{MLP}_{(\cdot)}$	Multilayer Perceptron.
$\sigma(\cdot)$	Sigmoid function.
$\mathbb{I}[\cdot]$	Indicator function
Meter	
$\widehat{T}_i$	Upper estimation of the input tokens of agent i.
$q_{\min}$	Minimum admissible completion length, see table 4.
$F_m(T)$	Floor cost of a call on backbone m.
R_i	Reserved floor cost for successors.
q_i	Completion cap of agent i.
$T_{\max}$	Maximum completion tokens per call, see table 4.
μ_i	Marginal price per output token of agent i.
ν_i	Whether a successor is still obliged.
κ_{r}	Output-to-downstream-input token factor, see table 4.
$\hat{\rho}_i$	Estimated input charge.
Stage 1: imitation	
$\mathcal{D}_{\text{SFT}}$	Demonstration set (train and val splits).
$\mathfrak{B}$	Log-spaced budget grid.
$\mathcal{R}_1(\mathcal{Q})$	Roles the policy can produce for the first agent.
K	Runs per template and query.
$\hat{u}_t,\ \hat{s}_t(q),\ \hat{c}_t$	Empirical utility, mean success of template t, and mean cost of template t.
$t^\star$	Selected template.
$\tau_i^\star,\ r_i^\star,\ e_{ji}^\star$	Logged target decisions.
$\lambda_{\text{stop}}, \lambda_{\text{role}}, \lambda_{\text{edge}}$	Loss weights.
w_r	Final-role weight of role r.
$\mathcal{S}_r,\ \mathcal{Q}_r$	Runs with final role r; their queries.
$\mathcal{L}_{\text{SFT}},\ \mathcal{L}_{\text{val}}$	Imitation loss; validation loss.
$\Delta_r,\ \tilde{\Delta}_r$	Final-role effect; its shrunk version.
$\zeta,\ [w_-, w_+]$	Temperature and clipping range of w_r.
$[\text{lo}_r, \text{hi}_r]$	95% bootstrap interval of Δ_r.
$E,\ P$	Epochs; patience.
N_e	Number of agents in demonstration e.
Z	Mean number of decisions per demonstration.
Stage 2: reinforcement learning	
G	Group size.
θ_{old}	Parameters of the sampling policy.
$U_g,\ A_g$	Utility and advantage of trajectory ξ_g.

Continued on next page

Symbol	Description
$\mathcal{K}(\xi),\ d$	Policy decisions of ξ; a decision.
$\rho_d(\theta)$	Likelihood ratio of decision d.
ϵ	Clipping range.
$\mathcal{L}_{\mathrm{RL}}$	Clipped surrogate loss.
$N_{\mathrm{rd}},\ E,\ \beta$	Rounds; epochs per round; minibatch.

B Supplementary Functions

History embedding.

$$f_{\mathrm{hist}}^{(i)} = \mathrm{GRU}\Big(\big[[z_{r_1}, z_{m_1}], \ldots, [z_{r_{i-1}}, z_{m_{i-1}}]\big]\Big),\ \ f_{\mathrm{hist}}^{(1)} = \mathbf{0}. \tag{10}$$

The GRU reads the role and backbone embeddings of the agents recruited so far, in generation order, and summarizes the team built up to step i. It is zero before the first agent is generated.

Connectivity of the previous agent.

$$f_{\mathrm{edge}}^{(i)} = \mathrm{MLP}_{\mathrm{struct}}\left([e_{1,i-1}, \ldots, e_{N_{\max},i-1},\ \mathbb{I}[1 < i{-}1], \ldots, \mathbb{I}[N_{\max} < i{-}1]]\right),\ \ \forall j \geq i{-}1\ e_{j,i-1} := 0, \tag{11}$$

The first $N_{\max}$ entries are the incoming edges $\mathbf{e}_{\cdot i-1}$ of the previous agent v_{i-1}, and the last $N_{\max}$ entries mark which agents it scanned as candidate predecessors, namely $v_1, \ldots, v_{i-2}$. Padding with zeros gives the input a fixed length $2N_{\max}$ at every step. It is the zero vector for $i \leq 2$, since v_1 has no predecessors.

Budget position.

$$\psi(x) = \mathrm{clip}\left(\frac{2\big(\log \max\{x,\ 10^{-3}B_{\min}\} - \log B_{\min}\big)}{\log B_{\max} - \log B_{\min}} - 1,\ -3,\ 1\right), \qquad x \in \{B,\ b_i\}. \tag{12}$$

The same function places the assigned budget B and the remainder b_i on one log-dollar scale. The training range $[B_{\min}, B_{\max}]$ maps linearly in $\log v$ onto $[-1, 1]$. Below $B_{\min}$ the scale extends down to -3, which lets a nearly exhausted remainder stay distinguishable. The floor $10^{-3}B_{\min}$ keeps the logarithm finite when b_i reaches zero or turns negative after an overrun.

Length vector.

$$\boldsymbol{\ell}_{j\to i} = \big(\log(1 + |y_j|_{\mathrm{chars}}),\ \log(1 + T_j^{\mathrm{out}}),\ \log(1 + T_j^{\mathrm{in}}),\ \log c_{m_i}^{\mathrm{in}}\big). \tag{13}$$

When the edge head decides whether v_i should read v_j, it sees how long v_j's response turned out to be, how many tokens v_j consumed and produced, and the input price of the receiving backbone m_i. Together these describe what connecting v_j would cost v_i to read. The first three entries come from the observation o_j and are known only after v_j has been executed. Logarithms compress lengths that vary over orders of magnitude.

Upper estimate of Input. $\widehat{T}_i = \lceil |q|_{\mathrm{B}}/\kappa_{\mathrm{b}} \rceil + \kappa_{\mathrm{p}} + \sum_{j<i} \big(\lceil |\tilde{y}_j|_{\mathrm{B}}/\kappa_{\mathrm{b}} \rceil + \kappa_{\mathrm{m}}\big)$ be an upper estimate of the input node i would consume if it read every completed predecessor.

Reserve. $R_i = \nu_i\, F_{\underline{m}}\big(\widehat{T}_i^{\mathrm{res}}\big)$ where $\nu_i = \mathbf{1}[i]$ the system would not satisfy the set of nodes with no outgoing edge after node i and $\widehat{T}_i^{\mathrm{res}}$ is the input the obliged successor would consume. Under $N_{\min} = 1$ the reserve vanishes in the mask of Eq. 7: with at least one agent already executed no successor is structurally required before the mask is consulted. It binds in the completion cap of Eq. 8, where a node that leaves two sinks obliges a merge.

Estimated Input Charge. $\hat{\rho}_i = c_{m_i}^{\mathrm{in}}\big(\lceil |x_i|_{\mathrm{B}}/\kappa_{\mathrm{b}} \rceil + \kappa_{\mathrm{c}}\big)$ the estimated input charge.

Table 4: Estimator constants and training hyperparameters. $\kappa_{\mathrm{r}} = \beta_{\mathrm{out}}/\kappa_{\mathrm{b}}$ with $\beta_{\mathrm{out}} = 8$ reserved output bytes per token. Choosing $\kappa_{\mathrm{b}} = 2$ rather than the customary 4 makes the input estimate roughly twice the expected token count; the guards are intended to be conservative, and the cost of that choice is a slightly smaller usable completion cap. The imitation weights λ are fixed in the SFT stage rather than read from the configuration.

Symbol	Role	Value	Config key
Estimator			
κ_{b}	UTF-8 bytes per estimated input token	2	`input_bytes_per_token`
κ_{p}	prompt reserve added to every mask estimate (tokens)	1664	`prompt_reserve_tokens`
κ_{c}	chat-format overhead added to every request estimate	768	`chat_overhead_tokens`
κ_{m}	per-predecessor margin in the mask estimate (tokens)	256	—
κ_{r}	input tokens a downstream reader pays per output token	4	derived
$q_{\min}$	minimum admissible completion	256	`min_completion_tokens`
$T_{\max}$	provider-side completion cap	32768	`max_tokens`
Objective			
γ	weight of the budget share in the utility	0.05	`budget_share_weight`
$\bar{\varepsilon}$	cap on the overrun ratio	1	`excess_cap`
$B_{\min}$	lower end of the budget range ($)	10^{-3}	`budget_min`
$B_{\max}$	upper end of the budget range ($)	5×10^{-2}	`budget_max`
Training			
λ_{stop}	weight of the termination term in $\mathcal{L}_{\mathrm{SFT}}$	0.2	—
λ_{role}	weight of the role term in $\mathcal{L}_{\mathrm{SFT}}$	0.2	—
λ_{edge}	weight of the edge term in $\mathcal{L}_{\mathrm{SFT}}$	0.8	—
G	trajectories per query–budget group	8	`group_size`
ϵ	clipping radius of the ratio in $\mathcal{L}_{\mathrm{RL}}$	0.2	`clip_epsilon`

C ALGORITHM WORKFLOW OF METERROUTER

Algorithm 1 Construction of $\mathcal{D}_{\mathrm{SFT}}$

Require: queries $\mathcal{D}$, templates $\mathcal{T}$, runs K, budget grid $\mathfrak{B}$, mid-priced backbone $\bar{m}$

$\mathcal{D}_{\mathrm{SFT}} \leftarrow \emptyset$
for $\mathcal{Q} \in \mathcal{D}$ **do**
  for $t \in \mathcal{T}$, $k = 1, \ldots, K$ **do**
    $r_i \sim \mathrm{Unif}(\mathcal{R})$, $m_i \leftarrow \bar{m}$ for every agent v_i of t
    $\xi_{t,k} \leftarrow \textsc{Execute}(t, \{r_i, m_i\}; \mathcal{Q}, B_{\max})$
  $\hat{c}_t \leftarrow \frac{1}{K}\sum_k C(\xi_{t,k}),\ t \in \mathcal{T}$
  for $B \in \mathfrak{B}$ **do**
    $t^\star \leftarrow \arg\max_{t:\,\hat{c}_t \le B}\ \hat{u}_t(\mathcal{Q}, B) = \frac{1}{K}\sum_{k=1}^{K} S(\mathcal{Q}, \xi_{t,k}) - \frac{\gamma}{B} \cdot \frac{1}{K}\sum_{k=1}^{K} C(\xi_{t,k})$
    for k with $S(\mathcal{Q}, \xi_{t^\star,k}) = 1$ **do**
      $e \leftarrow \big(\mathcal{Q}, B, \mathcal{G}(\xi_{t^\star,k}), o_{1:N}\big),\ b_i \leftarrow B - \sum_{j<i} C_j$ ▷ no re-execution
      if $m_i \in \mathcal{M}_i^m\ \forall i$ **then** $\mathcal{D}_{\mathrm{SFT}} \leftarrow \mathcal{D}_{\mathrm{SFT}} \cup \{e\}$
return $\mathcal{D}_{\mathrm{SFT}}$

Algorithm 2 Stage 1: budget-conditioned imitation

Require: $\mathcal{D}^{\text{train}}_{\text{SFT}}$, $\mathcal{D}^{\text{val}}_{\text{SFT}}$ (Algorithm 1), role weights $\{w_r\}$, epochs $E_{\max}$, patience P
1: freeze $\text{MLP}^{\text{GELU}}_{\text{route}}$ and MLP_B
2: **for** epoch $= 1, \dots, E_{\max}$ **do**
3: **for** minibatch $\beta \subset \{e_q \sim \text{Unif}(\mathcal{D}^{\text{train}}_{\text{SFT}}(q))\}_q$ **do**
4: $\theta \leftarrow \text{Adam}\big(\theta, \nabla_\theta \hat{\mathcal{L}}_{\text{SFT}}(\theta; \beta)\big)$ ▷ Eq. 14
5: **if** $\mathcal{L}_{\text{val}}(\theta)$ is the best so far **then** $\theta_{\text{SFT}} \leftarrow \theta$
6: **if** no improvement for P epochs **then break**
7: $Z \leftarrow \text{mean}_e \big(3N_e - 1 + N_e(N_e-1)/2 + \mathbb{I}[N_e < N_{\max}]\big)$
8: **return** θ_{SFT}, Z

Here, let $\mathcal{D}^{\text{train}}_{\text{SFT}}(q)$ be the demonstrations of query q and $\mathcal{Q}^{\text{SFT}} = \{q : \mathcal{D}^{\text{train}}_{\text{SFT}}(q) \neq \emptyset\}$.The imitation objective is:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\,\mathbb{E}_{q\sim\text{Unif}(\mathcal{Q}^{\text{SFT}})}\,\mathbb{E}_{e\sim\text{Unif}(\mathcal{D}^{\text{train}}_{\text{SFT}}(q))}\Bigg[\lambda_{\mathsf{stop}}\sum_{i\in\mathcal{I}} I^{\tau}_i \log\pi_\theta(\tau^\star_i \mid s_i) + \lambda_{\mathsf{role}}\sum_{i=1}^{N} I^{r}_i\, w_{r_N}^{\mathbb{I}[i=N]} \log\pi_\theta(r^\star_i \mid s_i) + \lambda_{\mathsf{edge}}\sum_{i=1}^{N}\sum_{j<i} I^{e}_{ji} \log\pi_\theta\big(e^\star_{ji} \mid r_i, m_i, \mathbf{e}_{>j,i}, s_i\big)\Bigg], \tag{14}$$

where $I^r_i = \mathbb{I}[i \geq 2 \ \vee \ r_1 \in \mathcal{R}_1(\mathcal{Q})]$ drops a logged first role that π_θ cannot produce, and w_r is the final-role weight of Eq. 16. The backbone factor of Eq. 2 is absent because $\text{MLP}^{\text{GELU}}_{\text{route}}$ is frozen. The two expectations are the sampling scheme of algorithm 2: every query receives equal weight, however many budgets and runs it contributes.

Early stopping monitors the structural imitation loss on the validation demonstrations:

$$\mathcal{L}_{\text{val}}(\theta) = -\,\frac{1}{|\mathcal{D}^{\text{val}}_{\text{SFT}}|}\sum_{e\in\mathcal{D}^{\text{val}}_{\text{SFT}}}\Bigg[\lambda_{\mathsf{stop}}\sum_{i\in\mathcal{I}} I^{\tau}_i \log\pi_\theta(\tau^\star_i \mid s_i) + \lambda_{\mathsf{edge}}\sum_{i=1}^{N}\sum_{j<i} I^{e}_{ji} \log\pi_\theta\big(e^\star_{ji} \mid r_i, m_i, \mathbf{e}_{>j,i}, s_i\big)\Bigg]. \tag{15}$$

Final-role weights. Within a query and a template, the K runs differ only in their randomly drawn roles, so the collected runs contain a paired contrast between roles. For every training run $\xi_{q,t,k}$, let $\hat{s}_t(q) = \frac{1}{K}\sum_{k'} S(\mathcal{Q}_q, \xi_{q,t,k'})$, $u_{q,t,k} = 2\big(S(\mathcal{Q}_q, \xi_{q,t,k}) - \hat{s}_t(q)\big)$, and let $r_N(\xi)$ be the role of the final agent of ξ. For each role r, let $\mathcal{S}_r = \{(q,t,k) : r_N(\xi_{q,t,k}) = r\}$ and let $\mathcal{Q}_r$ be the queries appearing in $\mathcal{S}_r$. Then:

$$\Delta_r = \frac{1}{|\mathcal{S}_r|}\sum_{(q,t,k)\in\mathcal{S}_r} u_{q,t,k}, \qquad \tilde{\Delta}_r = \text{sgn}(\Delta_r)\Big(|\Delta_r| - \frac{\text{hi}_r - \text{lo}_r}{2}\Big)_+,$$
$$w_r = \frac{\text{clip}\left(e^{\tilde{\Delta}_r/\zeta}, w_-, w_+\right)}{\text{mean}_{e\in\mathcal{D}^{\text{train}}_{\text{SFT}}}\text{clip}\left(e^{\tilde{\Delta}_{r_N(e)}/\zeta}, w_-, w_+\right)}, \tag{16}$$

where $[\text{lo}_r, \text{hi}_r]$ is the 95% interval of Δ_r from a bootstrap that resamples the queries in $\mathcal{Q}_r$ with replacement, and $\tilde{\Delta}_r = 0$ when $|\mathcal{Q}_r| < 2$. With $K = 2$, $u_{q,t,k} = S_k - S_{k'} \in \{-1, 0, 1\}$ compares the two runs of the same query and topology. Shrinking by the half-width of the interval keeps the unnormalized weight at one unless the effect of the final role exceeds its sampling uncertainty, and the normalization makes the average weight over the demonstrations equal to one. We use $\zeta = 0.1$ and $[w_-, w_+] = [0.25, 4]$.

Algorithm 3 Stage 2: group-relative reinforcement learning

Require: θ_{SFT}, Z (Algorithm 2), queries $\mathcal{D}$, group size G, rounds N_{rd}, epochs E

$\theta \leftarrow \theta_{\mathrm{SFT}}$
for $n = 1, \ldots, N_{\mathrm{rd}}$ **do**
  $\theta_{\mathrm{old}} \leftarrow \theta$
  for $\mathcal{Q} \in \mathcal{D}_n$ **do** ▷ queries of round n
    $B \sim \mathrm{LogUnif}[B_{\min}, B_{\max}]$
    $\xi_g \sim \pi_{\theta_{\mathrm{old}}}(\cdot \mid \mathcal{Q}, B),\ g = 1, \ldots, G$ ▷ Algorithm 4
    $A_g \leftarrow U(\mathcal{Q}, B, \xi_g) - \frac{1}{G}\sum_{g'} U(\mathcal{Q}, B, \xi_{g'})$
  for epoch $= 1, \ldots, E$; minibatch β of trajectories **do**
    $\rho_d(\theta) \leftarrow \pi_\theta(d)/\pi_{\theta_{\mathrm{old}}}(d),\ d \in \mathcal{K}(\xi), \xi \in \beta$
    $\theta \leftarrow \mathrm{Adam}\big(\theta, \nabla_\theta \hat{\mathcal{L}}_{\mathrm{RL}}(\theta; \beta)\big)$ ▷ Eq. 9
return θ

Algorithm 4 Budget-aware inference

Require: query $\mathcal{Q}$, budget B, trained policy π_θ

$b_1 \leftarrow B, \quad \mathcal{G} \leftarrow \emptyset$
for $i = 1, \ldots, N_{\max} + 1$ **do**
  if $i \geq 2$ **and** $\mathcal{G}_{<i}$ is connected **then**
    if $i = N_{\max} + 1$ **or** $\mathcal{M}_i^m = \emptyset$ **then break**
    $\tau_i \sim \pi_\theta(\tau_i \mid s_i)$
    if $\tau_i = 1$ **then break**
  $r_i \sim \pi_\theta(r_i \mid s_i)$
  $m_i \sim \pi_\theta(m_i \mid r_i, s_i)$ over $\mathcal{M}_i^m$
  for $j = i - 1, \ldots, 1$ **do**
    $e_{ji} \sim \pi_\theta(e_{ji} \mid r_i, m_i, \mathbf{e}_{>j,i}, s_i)$
  run v_i on m_i, reading $\{\tilde{y}_j : e_{ji} = 1\}$
  observe o_i; $\quad b_{i+1} \leftarrow b_i - C_i$; $\quad s_{i+1} \leftarrow (\mathcal{Q}, B, b_{i+1}, \mathcal{G}_{<i+1}, o_{\leq i})$
return the response y_N of the final agent v_N

D Supplementary Details of Experiments

We conclude the dataset statistics in table 5

Table 5: Dataset split used for training, validation, and testing.

Dataset	Training	Validation	Test
OmniMath-2	200	40	1,000
MMLU-Pro	200	40	1,000
MuSR	200	40	516
BBEH	200	40	1,000
MBPP+	200	40	138
Total	**1,000**	**200**	**3,654**

Table 6: LLM pool with each model's price tier, parameter counts, and context length. Prices are the accounting rates used to compute costs, taken from Artificial Analysis Artificial Analysis (2026).

LLM	Tier	Context	Price ($/1M tokens)	
			Input	Output
gpt-oss-20b	Low-cost	128K	0.06	0.19
gpt-oss-120b	Mid-cost	128K	0.15	0.595
MiniMax-M3 (∼428B)	High-cost	1M	0.30	1.20

E Supplementary Details of MeterRouter

Table 7: Templates in the predefined pool $\mathcal{T}$ used to build $\mathcal{D}_{\mathrm{SFT}}$. Templates below the rule are added for $N_{\max} = 5$.

Template	Agents	Edges	Parents per node
single_1	1	0	[]
chain_2	2	1	[], [0]
chain_3	3	2	[], [0], [1]
fan-in_3	3	2	[], [], [0, 1]
dense_3	3	3	[], [0], [0, 1]
chain_4	4	3	[], [0], [1], [2]
fan-in_4	4	3	[], [], [], [0, 1, 2]
diamond_4	4	4	[], [0], [0], [1, 2]
dense_4	4	6	[], [0], [0, 1], [0, 1, 2]
pruned_4	4	3–5	Random per query
chain_5	5	4	[], [0], [1], [2], [3]
fan-in_5	5	4	[], [], [], [], [0, 1, 2, 3]
dense_5	5	10	[], [0], [0, 1], [0, 1, 2], [0, 1, 2, 3]
pruned_5	5	5–9	Random per query

F Module Profile

Each dataset in our experiments has its own role profile, written for that dataset's domain in the style of ARG-Designer Li et al. (2026a). All role profiles are summarized below.

Role profile Omni-MATH-2

```
role_profile = [
{
    'Name': 'Math Solver',
    'Description': A math expert who solves the problem step by step,
      using other agents' answers only as hints.,
    'Prompt': 'You are a math expert. You will be given a math
     problem and hints from other agents. Give your own solving
     process step by step based on hints.'
    },
{
    'Name': 'Mathematical Analyst',
    'Description': 'A mathematical analyst who first analyzes the
     solution symbolically with letters, then substitutes values to
     calculate.',
    'Prompt': 'You are a mathematical analyst. You will be given a
     math problem, analysis and code from other agents. You need to
     first analyze the problem-solving process step by step, where
```

```
     the variables are represented by letters. Then you substitute
     the values into the analysis process to perform calculations and
      get the results.'
},
{
    'Name': 'Programming Expert',
    'Description': 'A programming expert who integrates step-by-step
     reasoning and Python code, which is executed to obtain the
     answer.',
    'Prompt': 'You are a programming expert. You will be given a math
      problem, analysis and code from other agents. Integrate step-by
     -step reasoning and Python code to solve math problems. Analyze
     the question and write code to solve the problem, using comments
      for the reasoning.',
},
{
    'Name': 'Inspector',
    'Description': 'An inspector who checks the logic, calculations,
     and code of other agents and gives a verified solution.',
    'Prompt': 'You are an Inspector. You will be given a math problem
     , analysis and code from other agents. Check whether the logic/
     calculation of the problem solving and analysis process is
     correct (if present). Check whether the code corresponds to the
     solution analysis (if present). Give your own solving process
     step by step based on hints.'
}
]
```

Role profile MBPP+

```
role_profile = [
{
    'Name': 'Project Manager',
    'Description': 'A project manager who oversees the overall code
     structure and suggests a concise, maintainable design.',
    'Prompt': 'You are a project manager. You will be given a
     function description and an example test by the user. You are
     responsible for overseeing the overall structure of the code,
     ensuring that the code is structured to complete the task.
     Implement code concisely and correctly without pursuing over-
     engineering. You need to suggest optimal design patterns to
     ensure that the code follows best practices for maintainability
     and flexibility. You can specify the overall design of the code,
      including the classes that need to be defined (maybe none) and
     the functions used (maybe only one function).'
},
{
    'Name': 'Algorithm Designer',
    'Description': 'An algorithm designer who specifies the algorithm
      and, for complex logic, its pseudocode.',
    'Prompt': 'You are an algorithm designer. You will be given a
     function description and an example test by the user. You need
     to specify the specific design of the algorithm, including the
     classes that may be defined and the functions used. When the
     implementation logic is complex, you can give the pseudocode
     logic of the main algorithm.'
},
{
    'Name': 'Programming Expert',
```

```
    'Description': A programming expert who writes the full
     implementation, considering other agents' code and test results
     .,
    'Prompt': 'You are a programming expert. You will be given a
     function description and an example test by the user. You may be
      able to get the output results of other agents. They may have
     passed the example test, but they may not be completely correct.
      Write your full implementation (restate the function signature)
     . Do not include anything other than a Python code block in your
      response. Do not change function names and input variable types
      in tasks.'
},
{
    'Name': 'Test Analyst',
    'Description': 'A test analyst who identifies problems in current
      code, special cases, and boundary conditions.',
    'Prompt': 'You are a test analyst. You will be given a function
     description and an example test by the user. You need to provide
      problems in the current code or solution based on the test data
      and possible test feedback in the question. You need to provide
      additional special use cases, boundary conditions, etc. that
     should be paid attention to when writing code. You can point out
      any potential errors in the code.'
},
{
    'Name': 'Bug Fixer',
    'Description': 'A bug fixer who provides modified and improved
     code based on the design, implementation, and test problems.',
    'Prompt': 'You are a bug fixer. You will be given a function
     description and an example test by the user. You need to provide
      modified and improved python code based on the current overall
     code design, algorithm framework, code implementation or test
     problems. Write your full implementation (restate the function
     signature). Do not include anything other than a Python code
     block in your response. Do not change function names and input
     variable types in tasks.'
},
{
    'Name': 'Normal Programmer',
    'Description': A programmer who writes a full implementation,
     referring to other agents' outputs.,
    'Prompt': You are a programmer. You will be given a function
     description and an example test by the user. You can refer to
     the agents' outputs. Write your full implementation (restate the
      function signature). Do not include anything other than a
     Python code block in your response.
}
]
```

Role profile MMLU-Pro

```
role_profile = [
{
    'Name': 'Knowledgeable Expert',
    'Description': 'A knowledgeable expert in question answering who
     identifies the key entities and the facts needed to answer.',
    'Prompt': 'You are a knowledgeable expert in question answering.
     Identify the key entities and concepts that the question depends
      on, and state the relevant facts you know about them before
     choosing an answer.'
},
```

```
{
    'Name': 'Critic',
    'Description': An excellent critic who points out potential
     issues in other agents' analysis point by point.,
    'Prompt': You are an excellent critic. Please point out potential
      issues in other agent's analysis point by point.
},
{
    'Name': 'Mathematician',
    'Description': 'A mathematician good at math games, arithmetic
     calculation, and long-term planning.',
    'Prompt': 'You are a mathematician who is good at math games,
     arithmetic calculation, and long-term planning.'
},
{
    'Name': 'Psychologist',
    'Description': 'A psychologist good at psychology, sociology, and
      philosophy.',
    'Prompt': 'You are a psychologist. You are good at psychology,
     sociology, and philosophy, and you reason scientifically about
     human behavior.'
},
{
    'Name': 'Historian',
    'Description': 'A historian who researches and analyzes cultural,
      economic, political, and social events in the past.',
    'Prompt': 'You research and analyze cultural, economic, political
     , and social events in the past, collect data from primary
     sources and use it to develop theories about what happened
     during various periods of history.'
},
{
    'Name': 'Doctor',
    'Description': 'A doctor with knowledge of medicine, physiology,
     and health.',
    'Prompt': You are a doctor. You are knowledgeable about medicine,
      physiology, diseases, and their treatment, and you consider a
     patient's age, lifestyle, and medical history when reasoning
     about health questions.
},
{
    'Name': 'Lawyer',
    'Description': 'A lawyer good at law, politics, and history.',
    'Prompt': 'You are good at law, politics, and history.'
},
{
    'Name': 'Economist',
    'Description': 'An economist good at economics, finance, and
     business.',
    'Prompt': 'You are good at economics, finance, and business. You
     have experience on understanding charts while interpreting the
     macroeconomic environment prevailing across world economies.'
},
{
    'Name': 'Programmer',
    'Description': 'A programmer good at computer science,
     engineering, and physics.',
    'Prompt': 'You are good at computer science, engineering, and
     physics. You have experience in designing and developing
     computer software and hardware.'
}
]
```

Role profile MuSR

```
role_profile = [
{
    'Name': 'Evidence Tracker',
    'Description': An evidence tracker who lists the story's facts
     about means, motive, opportunity, and alibis, separating stated
     facts from assumptions.,
    'Prompt': 'You are an evidence tracker. You will be given a long
     story, a question with options, and outputs from other agents.
     Extract the facts in the story that bear on the question, such
     as means, motive, opportunity, alibis, and contradictions.
     Separate what the story states from what is assumed, then weigh
     the evidence for each option step by step.'
},
{
    'Name': 'Timeline and Location Analyst',
    'Description': 'A timeline and location analyst who tracks where
     objects and people are over time and who saw each move.',
    'Prompt': You are a timeline and location analyst. You will be
     given a long story, a question with options, and outputs from
     other agents. Track step by step where each object and person is
      over time, who moved what, and who witnessed or missed each
     move, so that each character's beliefs about locations are clear
     . Then answer the question.
},
{
    'Name': 'Constraint Planner',
    'Description': A constraint planner who turns people's skills and
      relationships into constraints and evaluates each assignment.,
    'Prompt': You are a constraint planner. You will be given a long
     story, a question with options, and outputs from other agents.
     List each person's relevant skills, weaknesses, and working
     relationships as constraints, then evaluate every option against
      those constraints step by step and choose the assignment that
     best satisfies them.
},
{
    'Name': 'Critic',
    'Description': An excellent critic who points out unsupported
     inferences and overlooked story details in other agents'
     analysis.,
    'Prompt': You are an excellent critic. You will be given a long
     story, a question with options, and outputs from other agents.
     Please point out unsupported inferences, overlooked story
     details, and contradictions in other agents' analysis point by
     point.
},
{
    'Name': 'Inspector',
    'Description': An inspector who checks each option against the
     story's evidence and other agents' reasoning, then gives a
     verified answer.,
    'Prompt': You are an Inspector. You will be given a long story, a
      question with options, and outputs from other agents. Check
     whether other agents' reasoning is supported by the story, check
      each option against the decisive evidence, and give your own
     verified reasoning step by step.
}
]
```

Role profile BBEH

```
role_profile = [
{
    'Name': 'Math Solver',
    'Description': A solver who works through the task step by step,
     using other agents' answers only as hints.,
    'Prompt': 'You are a math expert. You will be given a hard
     reasoning task and hints from other agents. Give your own
     solving process step by step based on hints.'
},
{
    'Name': 'Mathematical Analyst',
    'Description': A mathematical analyst who analyzes the task's
     rules symbolically, then substitutes values to calculate step by
      step.,
    'Prompt': You are a mathematical analyst. You will be given a
     hard reasoning task and outputs from other agents. You need to
     first analyze the task's rules and the problem-solving process
     step by step, where unknown quantities are represented by
     letters. Then you substitute the values into the analysis
     process to perform calculations and get the results.
},
{
    'Name': 'Programming Expert',
    'Description': A programming expert who encodes the task's rules
     in Python code, which is executed to obtain the answer.,
    'Prompt': You are a programming expert. You will be given a hard
     reasoning task and outputs from other agents. Encode the task's
     rules, operations, and data faithfully in Python and compute the
      result, using comments for the reasoning.
},
{
    'Name': 'Inspector',
    'Description': An inspector who checks whether other agents
     followed the task's exact rules and answer format, then gives a
     verified answer.,
    'Prompt': You are an Inspector. You will be given a hard
     reasoning task and outputs from other agents. Check whether the
     logic, calculations, and code of other agents follow the task's
     exact rules (if present), and whether their answers use the
     representation the task asks for. Give your own solving process
     step by step based on hints.
},
{
    'Name': 'Logician',
    'Description': 'A logician who formalizes clues and constraints,
     deduces consequences, and eliminates inconsistent possibilities
     .',
    'Prompt': 'You are a logician. You will be given a hard reasoning
      task and outputs from other agents. Formalize the clues,
     statements, and constraints, deduce their consequences step by
     step, and eliminate possibilities that contradict them until the
      answer is determined.'
},
{
    'Name': 'Linguist',
    'Description': 'A linguist who analyzes word order, meaning,
     ambiguity, humor, and unfamiliar language patterns.',
    'Prompt': 'You are a linguist. You will be given a hard reasoning
      task and outputs from other agents. Analyze the language in the
      task, including grammar, word order, word meanings, ambiguity,
```

```
      sarcasm or humor, and patterns in unfamiliar languages, and
      reason step by step to the answer.'
  },
  {
      'Name': 'Critic',
      'Description': An excellent critic who points out rule violations
        and reasoning errors in other agents' analysis.,
      'Prompt': You are an excellent critic. You will be given a hard
       reasoning task and outputs from other agents. Please point out
       rule violations, reasoning errors, and answer-format mistakes in
        other agents' analysis point by point.
  }
  ]
```

G CONFIGURATION OF BASELINE METHODS

Table 8: Experimental settings of the baselines. **Calls/Query** is the average number of LLM calls per evaluated question.

Method	Training	Structure	Calls/Query	Hyper-params.
Vanilla	None	Single LLM, zero-shot	1	–
CoT	None	Single LLM, 3-shot CoT	1	–
Chain	None	5 agents, chain	5	No pruning
Tree	None	5 agents, binary tree + sink aggregation	6	No pruning
Complete Graph	None	5 agents, complete DAG	5	No pruning
LLM-Debate	None	3 agents, 2 rounds, majority vote	6	No pruning
AutoGen	None	Round-robin team: solver, critic, solver	3	Max 3 turns
GPTSwarm	None	3 IO agents + final decision node	4	Fixed graph (no edge/ node optimization)
AgentPrune	Edge masks learned on the training split	5 agents, fully connected, + decision node; pruned	6	1 round (spatial edges only); 10 iters, batch 4, lr 0.1; prune 25% every 5 iters
GraphRouter	GNN router on MiniLM embeddings, 1000 epochs; labels from each LLM	One LLM per query	1	Performance First; batch 32, lr 3×10^{-4}
RouterDC	mDeBERTa-v3-base router, 1000 steps; labels from 10 samples/LLM (T=0.2)	One LLM per query	1	Batch 64, lr 5×10^{-5}; K^+=1, K^-=2[§2]; 5 clusters
BAMAS	Per-benchmark policy, 0.05 budget, 10 epochs	4 candidate topologies; ILP team of 5 (10 on Omni[§1] due to different topology)	3.9	Budget \$0.05/query; linear $\leq$ 5 steps
MasRouter	Per-benchmark controller, 5 epochs (10 on MBPP+)	Up to 6 agents, 6 collab. modes	9.0	Batch 16, lr 0.01; cost penalty 100 (Omni[§1]), 400 (MBPP+), 200 (rest)

[§1]Omni denotes Omni-Math-2. [§2]The paper uses $K^+ = K^- = 3$, which is degenerate with three candidate LLMs.