CO-EVOLVING QUERY ROUTING AND AGENT SPECIALIZATION IN MIXTURE-OF-AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Agents systems typically treat query routing and agent post-training as separate stages. A router fitted to a frozen pool cannot track later policy updates, and agents are not given competence-aware data that would induce complementary expertise. We study this coupling as co-evolution: a router and a population of independently updated LLM agents are optimized in the same reinforcement learning loop, so that allocation and specialization change together. Before generation, a predictive familiarity estimator maps intermediate hidden states to a familiarity score that estimates how well an agent matches a query, without full rollouts or external judges. Cumulative-threshold routing then activates the smallest score-ranked subset whose familiarity scores sum to at least a threshold, trading accuracy against compute. The same scores steer training queries toward agents that are becoming proficient on related inputs, shifting a pool of homogeneous generalists into specialists. Across multiple domains, the resulting system outperforms static-agent routers and fine-tuning pipelines that keep the collaboration workflow fixed.

1 INTRODUCTION

Although large language models (LLMs) have demonstrated remarkable reasoning capabilities, single-model post-training methods often saturates on mixed problem domains (Hong et al., 2024; Ye et al., 2025). To transcend the limitations of single models, the Mixture-of-Agents (MoA) paradigm has emerged as a promising solution. By combining multiple agents, this paradigm aims to solve diverse tasks more effectively by leveraging the complementary strengths of different agents (Wang et al., 2025a). Such collaborative frameworks have the potential to expand the performance boundaries of LLMs on complex tasks.

Despite this potential, the current development of multi-agent systems primarily splits into two directions. The first direction focuses on optimizing orchestration mechanisms among fixed-capability agents, which is typically achieved through debate frameworks to refine reasoning consensus (Chan et al., 2024; Estornell & Liu, 2024; Yi et al., 2025; Hu et al., 2026; Fan et al., 2026; Qiao et al., 2026), or through dynamic agent selection for efficient query allocation (Xia et al., 2024; Yue et al., 2025; Lee et al., 2026; Poon et al., 2026; Wang et al., 2026a; Xue et al., 2026a; Wang et al., 2026c). The second direction explores training and fine-tuning agents, but typically operates within predetermined multi-agent workflow architectures (Park et al., 2025; Motwani et al., 2025; Zhang et al., 2025; Xue et al., 2026b; Zhao et al., 2026; Wang et al., 2026d). Although these approaches effectively enhance collective performance, they inherently decouple the routing strategy from the continual learning dynamics of agents.

This decoupling exposes a critical research gap in multi-agent paradigms. First, the capabilities of individual agents continually evolve during the post-training phase. However, existing routing mechanisms are not designed to adapt to such capability shifts. Second, current post-training pipelines typically rely on manually partitioned datasets, lacking a dynamic sample allocation mechanism. Without routing specific training queries to agents based on their competence, the system fails to automatically induce distinct, targeted expertise. Consequently, the disconnect between routing strategies and the continual learning of agents prevents the system from achieving synergistic capability specialization, leaving individual agents acting as generalists rather than domain experts.

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

To bridge this critical disconnect, we formulate Co-Evolving Router with continually learning Agents for Mixture-of-Agents (CERA-MoA), which co-evolves a router with independently updated LLM agents in the same reinforcement-learning cycle. The router follows a reverse Random Network Distillation (RND) design (Burda et al., 2018): a frozen language-model encoder extracts mid-layer hidden states of the last non-padding token of the query, and each agent is paired with a trainable predictor head and a frozen random target head. Standard RND treats predictor–target error as exploration novelty; we invert that geometry and map the distance to a familiarity score, i.e., how well a query currently aligns with an agent. This estimator evaluates relative competence before generation, avoiding the cost of external judges or full candidate rollouts. Utilizing these familiarity scores, we further introduce a cumulative-threshold adaptive routing strategy. Rather than relying on a fixed top-k allocation, the router activates the smallest familiarity-ranked subset of agents whose cumulative score exceeds a threshold. Queries that are already well matched by a few specialists therefore incur little compute, while queries whose cumulative familiarity is insufficient recruit additional agents, yielding a fine-grained trade-off between task performance and computational cost. The same rule also governs training: rather than treating routing and agent adaptation in isolation, CERA-MoA not only adapts to the progressively shifting capabilities of individual agents, but also allocates each training query to the agents with high familiarity, so they receive tailored samples and explicitly induce skill specialization.

Our contributions are: (i) a sample-level Mixture-of-Agents framework that trains independently updated LLM agents and a familiarity router from the same rewards, so allocation can track policy updates and in turn assign the data that drives specialization; (ii) a predictive familiarity estimator for pre-generation compatibility scoring, together with a cumulative-threshold routing rule that flexibly activates by a familiarity budget; and (iii) experiments on shared-backbone and heterogeneous pools across mathematical reasoning, code generation, instruction following, and general reasoning, showing consistent gains over static-agent routers and fixed-workflow fine-tuning.

2 Related Work

Multi-agent systems (MAS) have emerged as a powerful paradigm to extend the reasoning boundaries of large language models (LLMs) by leveraging collective intelligence (Zhao et al., 2024; Ye et al., 2025). To facilitate effective collaboration, standard MAS architectures typically organize models through structured interaction protocols, such as debate (Liang et al., 2024; Estornell & Liu, 2024), majority voting (Chen et al., 2024; Taubenfeld et al., 2025), or mixtures of independent agents (Zhao et al., 2024; Wang et al., 2024; Xie et al., 2025; Li et al., 2026b; Ye et al., 2025). Unlike token-level Mixture-of-Experts (MoE) architectures that route over sub-networks (Oldfield et al., 2024; Lv et al., 2025; Zhuang et al., 2025), MAS operates at the sample and semantic level, requiring high-level coordination among autonomous models. We group prior work into orchestration optimization of frozen agents and agent fine-tuning inside a fixed workflow.

Orchestration Optimization. Query routing has been studied with bandits (Xia et al., 2024; Poon et al., 2026), budgeted knapsacks (Wang et al., 2025b; Xue et al., 2026a), diversity-aware selection (Xie et al., 2025), lightweight scorers (Yue et al., 2025; Wang et al., 2026a;c), and confidence-guided stepwise routing (Lee et al., 2026; Wang et al., 2026b). Recently, studies have also explored leveraging LLMs directly as self-orchestrators (Dang et al., 2026; Ke et al., 2026), topology graph generators (Zhang et al., 2026; Li et al., 2026b; Feng et al., 2026), and meta-thinkers (Zhu et al., 2026), while methods like AgentDropout (Wang et al., 2025c) prune redundant nodes to reduce overhead. ReSo (Zhou et al., 2025) selects agents from a task graph with a collaborative reward model, and differentiable Mixture-of-Agents (Wu et al., 2026) routes frozen agents step by step at test time. These methods adapt *which frozen agents to call*. They do not update the called policies from the same loop that trains the router. CERA-MoA instead trains the router together with agent LoRA (or heterogeneous model) policies, so allocation can change after a policy update.

Agent Fine-tuning. Agents have been trained with self-verification (Lee et al., 2025), chain refinement (Puerto et al., 2025), and reflection cycles (Zhao et al., 2025; Yuan & Xie, 2025). Collaborative reinforcement learning uses debate rewards (Park et al., 2025), tree-structured sampling (Motwani et al., 2025; Zhao et al., 2026), LLM-as-a-judge loops (Xue et al., 2026b), and end-to-end multi-agent RL (Wang et al., 2026d). CoMAS (Xue et al., 2026b) co-evolves agents from interaction

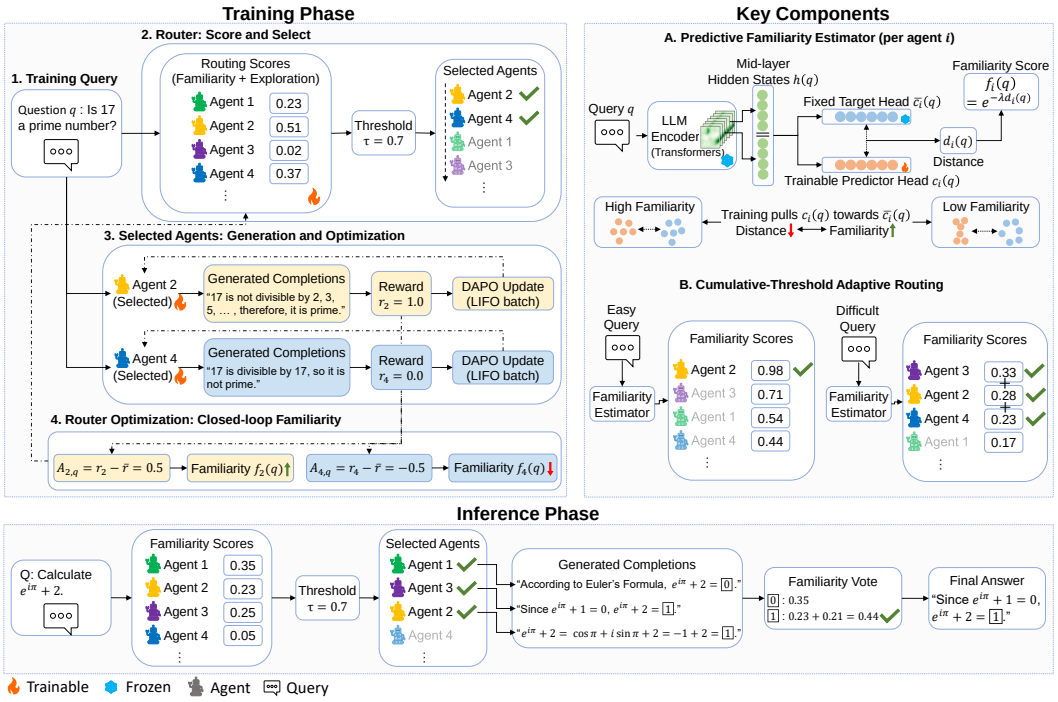


Figure 1: Overview of CERA-MoA.

rewards inside a judge-mediated workflow. HiVA (Tang et al., 2025) jointly evolves agent semantics and a routing topology. EvolveRouter (Huang et al., 2026) co-evolves routing with prompt revisions and stops by weighted agreement. MoRSE (Li et al., 2026a) learns role- and subtask-conditioned LoRA experts whose roles are specified in advance. Across these pipelines, either the agents stay frozen while the router changes, or the agents train while allocation stays a fixed workflow, prompt, or role graph. CERA-MoA makes familiarity-based routing part of the RL training loop: the router assigns data, and relative rewards update only the selected agents' heads.

3 CERA-MoA

3.1 Overview

As illustrated in Figure 1, we formulate CERA-MoA (Co-Evolving Router with continually learning Agents for Mixture-of-Agents) as a collaborative framework $\mathcal{M} = \left(\mathcal{D}_\psi, \{\mathcal{A}_{\theta_i}\}_{i=1}^N, \mathcal{S}\right)$, integrating a parameterized router $\mathcal{D}_\psi$, a population of continually learning agent policies $\{\mathcal{A}_{\theta_i}\}_{i=1}^N$, and a voting aggregator $\mathcal{S}$. Our primary design objective is to establish a closed-loop co-evolutionary process for the router and the agents: adaptive query allocation routes targeted training samples to specific agents to drive domain specialization, while the router synchronously updates its parameters to accurately track these continually shifting agent capabilities.

Training Phase. For an incoming query, the router $\mathcal{D}_\psi$ computes exploration terms and familiarity scores that measure relative agent competence, and allocates the query to a tailored subset of agents. Selected agents generate completions and optimize their policies through reinforcement learning, driven by task rewards. Concurrently, the router evaluates the relative advantages of agent performance to update its familiarity scorer. This iterative feedback loop naturally encourages specialization: agents receive problems aligned with their potential, evolving from homogeneous generalists into domain specialists, while the router synchronously tracks their real-time expertise. Unselected agents update neither their policy nor their predictor head on that query.

Inference Phase. $\mathcal{D}_\psi$ activates the smallest subset whose familiarity scores meet τ, while exploration terms are turned off. Each activated agent generates one response. For tasks with a deterministic extracted answer, $\mathcal{S}$ uses familiarity-weighted majority voting

$$\hat{y} = \arg\max_{y} \sum_{i \text{ s.t. } \mathrm{ans}(a_i)=y} f_i(q), \tag{1}$$

and returns the trajectory of the highest-f_i agent in the winning group. Here $\mathrm{ans}(\cdot)$ is the rule-based extractor in Appendix C (boxed-answer match, choice letter, or constraint-verifier output). For open-ended code, $\mathcal{S}$ returns the most familiar agent's completion without merging.

3.2 Predictive Familiarity Estimator

The estimator uses the same but inverted two-network geometry as Random Network Distillation (Burda et al., 2018): for each agent i, a trainable predictor head g_i is matched against a frozen target head $\bar{g}_i$, and closeness, rather than prediction error, is read as familiarity. Only g_i is trained, and the frozen target is not a learned model. Query embeddings come from a frozen language-model encoder: $h(q)$ concatenates the *last non-padding* hidden states at layers $\lfloor L/2 \rfloor$ and $\lfloor 3L/4 \rfloor$,

$$h(q) = \mathrm{concat}\left(\mathrm{hidden}_{\lfloor \frac{L}{2} \rfloor}(q) \middle\| \mathrm{hidden}_{\lfloor \frac{3L}{4} \rfloor}(q)\right). \tag{2}$$

Mid-layer states carry semantic structure that is less tied to next-token prediction than the final layer (Skean et al., 2025). Each head is an MLP applied to this shared embedding. Target heads are randomly initialized and never updated, so each agent has a fixed reference in the same feature space. With ℓ_2-normalized projections $c_i = g_i(h(q))$ and $\bar{c}_i = \bar{g}_i(h(q))$, the head distance is

$$d_i(q) = \left\| \frac{c_i(q)}{\|c_i(q)\|_2} - \frac{\bar{c}_i(q)}{\|\bar{c}_i(q)\|_2} \right\|_2 \in [0, 2]. \tag{3}$$

The familiarity score is

$$f_i(q) = \exp(-\lambda d_i(q)) \in \left[e^{-2\lambda}, 1\right]. \tag{4}$$

Each $d_i(q)$ is evaluated only inside the pair $(g_i, \bar{g}_i)$ on the same $h(q)$: frozen targets are not compared with one another, so the direction of $\bar{c}_i$ does not enter d_j for $j \neq i$. Across agents we compare only how close each predictor sits to its own target. Orthogonal initialization of the linear maps sets their singular values to one, so that on a shared $h(q)$ the heads start with matched Jacobian scales and comparable gradient norms, rather than letting a randomly scaled head dominate early updates. Because $\bar{g}_i$ is frozen, $d_i(q)$ moves only when the predictor g_i is updated. Positive relative advantages pull c_i toward this anchor around the query; negative advantages push it away up to a margin m. We set m so that $Ne^{-\lambda m} \leq \tau$: if every head sits at the margin, the sum cannot meet τ and the query falls back to the full pool.

3.3 Cumulative-Threshold Adaptive Routing

We introduce a familiarity-aware, cumulative-threshold allocation mechanism that balances reasoning efficacy against multi-agent inference overhead by activating the minimal agent subset required to reach a target competence threshold.

During training, routing scores incorporate exploration bonuses alongside familiarity scores to prevent starvation, as well as historical entropy to encourage policy diversity:

$$s_i(q) = f_i(q) + c_{ucb}\sqrt{\frac{\log T}{n_i + 1}} + w_{ent}\overline{H}_i, \tag{5}$$

where T is the number of routed queries, n_i is the number of training samples allocated to agent i, and $\overline{H}_i$ represents the agent's historical mean generation entropy. Let (j) denote the index of the agent ranked j-th in descending order by the routing score $s_i(q)$. Crucially, while the routing score $s_i(q)$ determines the priority of agent selection to foster exploration during training, the cutoff for subset activation is strictly evaluated against familiarity scores. Thus, the router activates the smallest top-ranked subset of agents whose cumulative familiarity score $f_{(j)}(q)$ satisfies:

$$\min_k \sum_{j=1}^{k} f_{(j)}(q) \geq \tau, \tag{6}$$

where τ is the predefined cumulative threshold. If the total sum is below τ, the query is allocated to the entire population. At inference time, the exploration terms are deactivated ($c_{ucb} = w_{ent} = 0$), strictly routing queries to the most competent specialists while preserving efficiency.

3.4 Joint Optimization of Router and Agents

The engine driving CERA-MoA is a closed-loop reinforcement learning protocol that bridges agent policy optimization with synchronous router updates. Selected agents store allocated queries in a LIFO buffer. When a micro-batch is full, agent i samples G completions and updates with Dynamic Sampling Policy Optimization (DAPO) (Yu et al., 2026) and the sequence-level importance sampling of Group Sequence Policy Optimization (GSPO) (Zheng et al., 2025).

Let $R\left(q, a_{i,q}^{(g)}\right)$ denote the reward assigned to the g-th completion of question q by agent i. The normalized advantage for the g-th completion is

$$\hat{A}_{i,q}^{(g)} = \frac{R\left(q, a_{i,q}^{(g)}\right) - \mu_{i,q}}{\sigma_{i,q} + 10^{-4}}, \tag{7}$$

where $\mu_{i,q}$ and $\sigma_{i,q}$ are the empirical mean and standard deviation of the G rewards.

For a completion $a_{i,q}^{(g)} = (y_1, \ldots, y_S)$, the sequence-level importance ratio is averaged over valid completion tokens:

$$\rho_{i,q}^{(g)} = \exp\left(\frac{1}{S}\sum_{t=1}^{S} \log \frac{\pi_{\theta_i}(y_t \mid q, y_{<t})}{\pi_{\theta_i^{\text{old}}}(y_t \mid q, y_{<t})}\right). \tag{8}$$

Then we define the clipped surrogate as

$$s_{i,q}^{(g)} = -\min\left(\rho_{i,q}^{(g)}\hat{A}_{i,q}^{(g)}, \text{clip}\left(\rho_{i,q}^{(g)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_{i,q}^{(g)}\right). \tag{9}$$

To keep the policy from drifting too far from a frozen reference π_{ref}, we add a token-level *k3* KL estimator. With

$$x_t = \log \pi_{\text{ref}}\left(y_t \mid q, y_{<t}\right) - \log \pi_{\theta_i}\left(y_t \mid q, y_{<t}\right), \tag{10}$$

the k3 estimate at token t is

$$D_{\text{KL},t} = \exp(x_t) - x_t - 1. \tag{11}$$

The loss for agent i is normalized by the number of active completion tokens in the accumulated training batch:

$$\mathcal{L}_{\text{agent}}(\theta_i) = \frac{\sum_{q,g,t} m_{q,g,t}\left(s_{i,q}^{(g)} + \beta D_{\text{KL},t}\right)}{\sum_{q,g,t} m_{q,g,t}}, \tag{12}$$

where $m_{q,g,t}$ masks padding tokens and β acts as the regularization scaling hyperparameter.

The router is updated from the same rewards, using only the agents that were activated on q. Let $\mathcal{S}_q$ be that activated subset, and write the mean completion reward of agent i as

$$r_{i,q} = \frac{1}{G}\sum_{g=1}^{G} R\left(q, a_{i,q}^{(g)}\right). \tag{13}$$

The cross-agent relative advantage is

$$A_{i,q} = r_{i,q} - \frac{1}{|\mathcal{S}_q|}\sum_{j \in \mathcal{S}_q} r_{j,q}. \tag{14}$$

If $|\mathcal{S}_q| = 1$, we set $A_{i,q} = r_{i,q}$. Minimizing

$$\mathcal{L}_{\text{router}}(g_i) = \sum_{q \in \mathcal{Q}} \begin{cases} A_{i,q} d_i(q), & \text{if } A_{i,q} \geq 0; \\ |A_{i,q}| \max(0, m - d_i(q)), & \text{if } A_{i,q} < 0, \end{cases} \tag{15}$$

pulls a relatively stronger agent toward its frozen target on q, raising future familiarity for similar queries, and pushes a relatively weaker agent away up to a margin m. The router therefore tracks who is currently better on which prompts, and the same familiarity scores continue to allocate training data, so agents that succeed in a neighborhood of queries receive more of those queries and specialize. Unselected agents receive neither completions nor this gradient on that query.

Table 1: In-distribution (ID) comparison across base models. Best is **bold**; second-best is underlined.

Method	Math			Code			Instr.		Gen.	Avg.
	GSM8k	MATH	DAPO	MBPP	Eurus	TACO	MAG.	IF	BBH	
Qwen3-4B	91.2	71.5	25.6	25.3	12.9	3.2	90.6	59.6	66.8	49.6
+ ICL-Router	91.6	75.4	30.0	25.7	12.7	3.0	90.2	59.8	67.6	50.7
+ LinUCB	92.0	71.6	25.2	23.7	13.0	3.3	91.2	59.4	70.6	50.0
+ RouteMoA	**93.5**	75.6	28.8	32.7	15.1	3.5	**92.8**	62.0	74.2	53.1
+ GSPO	91.4	76.1	35.6	59.9	27.3	12.5	91.0	64.2	76.6	59.4
+ MAPoRL	91.7	75.0	38.4	**64.2**	23.8	6.4	89.0	59.8	76.4	58.3
+ AT-GRPO	91.4	77.6	38.0	63.8	29.8	12.0	90.4	66.0	79.8	61.0
+ CERA-MoA	**93.5**	**79.6**	**41.8**	60.3	**31.3**	**13.4**	**92.8**	**72.4**	**83.4**	**63.2**
Llama-3.2-3B	62.9	34.6	10.4	41.2	4.9	0.8	66.4	41.0	39.8	33.6
+ ICL-Router	64.8	40.8	10.8	43.6	7.2	1.3	68.6	46.0	47.0	36.7
+ LinUCB	65.3	41.2	14.6	42.0	7.5	1.5	69.2	44.6	47.8	37.1
+ RouteMoA	79.2	47.3	16.2	**49.0**	9.7	1.3	79.0	50.8	57.4	43.3
+ GSPO	81.5	47.1	14.8	28.0	7.8	2.0	74.2	48.8	55.4	40.0
+ MAPoRL	81.3	49.1	14.8	40.9	6.8	1.1	73.6	43.2	57.8	41.0
+ AT-GRPO	80.8	48.8	16.2	25.3	7.9	**2.5**	74.2	51.6	59.4	40.7
+ CERA-MoA	**84.0**	**51.7**	**21.8**	44.7	**11.3**	2.0	**83.2**	**64.2**	**68.4**	**47.9**
Phi-4-mini	85.0	56.7	17.6	53.3	10.6	2.3	78.6	43.6	47.2	43.9
+ ICL-Router	87.6	62.6	22.8	52.5	11.5	2.9	81.0	43.4	54.2	46.5
+ LinUCB	86.9	63.0	20.6	51.4	14.9	3.0	82.0	43.4	54.2	46.6
+ RouteMoA	**91.5**	67.8	24.2	**60.7**	15.5	4.7	87.4	49.2	61.6	51.4
+ GSPO	91.0	65.4	20.6	56.4	**22.7**	8.3	87.2	56.2	73.4	53.5
+ MAPoRL	89.6	67.1	24.0	59.9	19.9	6.7	82.8	52.2	75.2	53.0
+ AT-GRPO	89.8	66.9	23.6	54.9	22.1	**8.4**	87.0	59.2	75.4	54.1
+ CERA-MoA	91.2	**68.6**	**26.8**	59.5	22.2	7.7	**87.6**	**61.0**	**76.0**	**55.6**
Heterogeneous										
+ ICL-Router	90.0	68.1	26.4	55.3	13.2	2.5	84.2	56.4	67.6	51.5
+ LinUCB	91.7	70.9	26.0	46.3	13.0	3.1	90.2	57.2	69.6	52.0
+ RouteMoA	93.1	68.7	25.2	**63.0**	16.9	4.0	90.4	56.6	75.2	54.8
+ MAPoRL	91.6	70.0	29.8	57.6	23.9	9.0	88.8	61.2	76.6	56.5
+ AT-GRPO	**93.9**	76.2	35.0	**63.0**	**30.1**	**13.1**	91.8	69.2	**84.0**	61.8
+ CERA-MoA	93.1	**81.5**	**44.2**	59.5	23.8	12.5	**92.8**	**75.4**	83.8	**63.0**

4 EXPERIMENTS

The empirical evaluation is designed to answer the following research questions:

RQ1: Does CERA-MoA outperform static-agent routing and fixed-workflow agent fine-tuning?

RQ2: How effectively does CERA-MoA adapt to a heterogeneous pool of base models?

RQ3: Is the predictive familiarity estimator a better routing metric than direct reward regression or multi-class classification?

RQ4: Can cumulative-threshold routing balance task performance against the number of activated agents?

RQ5: How does CERA-MoA induce distinct capability specialization among agents?

4.1 EXPERIMENTAL SETUP

Datasets. We evaluate across four domains: mathematical reasoning (GSM8k (Cobbe et al., 2021), Hendrycks MATH (Hendrycks et al., 2021), DAPO-MATH-17k (Yu et al., 2026)), code generation (MBPP (Austin et al., 2021), Eurus-2-Code (Yuan et al., 2024), TACO (Li et al., 2023)), instruction following (MAGPIE-IF (Xu et al., 2025), RLVR-IFEval (Lambert et al., 2024)), and general reasoning (BIG-bench Hard (Suzgun et al., 2023)). We train both the agents and the router on the training splits and report performance on independently partitioned test sets. Out-of-distribution

Table 2: Comparison of CERA-MoA with baselines on out-of-distribution (OOD) tasks.

Method	IFEval	HumanEval	AGIEval	ARC-c	LogicBench	Olympiad	Avg.
Qwen3-4B	79.3	36.4	64.0	91.7	17.0	29.7	53.0
+ ICL-Router	78.0	37.0	67.6	91.1	17.1	35.5	54.4
+ LinUCB	77.6	30.5	64.0	92.7	17.4	29.7	52.0
+ RouteMoA	**83.0**	44.2	67.5	**94.4**	25.7	32.5	57.9
+ GSPO	79.1	78.6	68.2	92.5	65.4	39.2	70.5
+ MAPoRL	78.7	**85.1**	66.4	93.3	65.5	34.2	70.5
+ AT-GRPO	78.6	84.4	69.1	93.2	64.8	38.9	71.5
+ CERA-MoA	**83.0**	82.5	**71.3**	92.7	**65.6**	**41.6**	**72.8**
Llama-3.2-3B-Instruct	62.3	48.7	29.7	71.6	37.2	8.2	43.0
+ ICL-Router	69.7	49.4	35.8	72.6	41.6	12.0	46.9
+ LinUCB	67.8	48.1	36.0	73.6	41.5	11.6	46.4
+ RouteMoA	69.7	61.7	43.4	79.6	**49.3**	13.7	52.9
+ GSPO	58.0	55.8	42.3	78.3	44.8	15.3	49.1
+ MAPoRL	53.0	51.9	41.8	79.2	44.9	13.7	47.4
+ AT-GRPO	71.0	50.0	41.6	78.7	**49.3**	15.6	51.0
+ CERA-MoA	**72.1**	**62.3**	**46.2**	**80.2**	44.9	**17.4**	**53.9**
Phi-4-mini-Instruct	69.9	57.8	50.5	71.2	18.0	24.3	48.6
+ ICL-Router	67.1	66.2	56.9	71.5	27.6	27.9	52.9
+ LinUCB	70.4	66.9	56.4	69.9	33.0	28.3	54.2
+ RouteMoA	70.2	79.9	59.9	82.8	27.1	**29.8**	58.3
+ GSPO	68.9	70.8	58.6	87.8	35.9	27.6	58.3
+ MAPoRL	60.6	**80.5**	58.8	88.1	34.1	28.3	58.4
+ AT-GRPO	73.9	74.0	59.6	88.4	35.3	29.0	60.0
+ CERA-MoA	**74.7**	77.3	**60.7**	**89.6**	**42.6**	**29.8**	**62.5**
Heterogeneous pool							
+ ICL-Router	74.9	50.0	60.1	89.3	18.9	31.0	54.0
+ LinUCB	77.6	52.6	64.0	92.3	17.2	30.8	55.8
+ RouteMoA	78.6	53.9	61.4	92.4	26.1	30.7	57.2
+ MAPoRL	68.9	78.6	62.3	89.6	54.5	31.8	64.3
+ AT-GRPO	81.3	**83.8**	67.8	**92.9**	50.9	37.6	69.1
+ CERA-MoA	**83.4**	77.3	**71.1**	92.7	**62.4**	**44.1**	**71.8**

(OOD) generalization is evaluated on IFEval (Zhou et al., 2023), HumanEval (Chen et al., 2021), AGIEval (Zhong et al., 2023), ARC-c (Clark et al., 2018), LogicBench (Parmar et al., 2024), and OlympiadBench (He et al., 2024), without additional fine-tuning.

Baselines. We compare CERA-MoA against static-agent orchestration and predetermined-workflow agent fine-tuning. Orchestration baselines include **ICL-Router** (Wang et al., 2026a), which constructs capability profiles in a reconstructed latent space; **LinUCB** (Poon et al., 2026), which treats query routing as a contextual bandit; and **RouteMoA** (Wang et al., 2026c), which uses a contrastive scorer to activate multiple expert models. Fine-tuning baselines include single-agent **GSPO** (Zheng et al., 2025); **MAPoRL** (Park et al., 2025), which trains agents inside a debate workflow with task and cross-agent correction rewards; and **AT-GRPO** (Zhao et al., 2026), which applies Group Relative Policy Optimization inside a predefined tree-structured MoA.

Implementation Settings. We study two kinds of agent pools. In the shared-backbone setting, $N = 4$ agents share one base model, each with an independent LoRA adapter (Hu et al., 2021). Orchestration baselines simulate four experts with role-specific prompts, so a 4-agent system on a 4B model occupies only $\sim$4.4B parameters. In the heterogeneous setting, $N = 3$ agents are distinct open-source models: Qwen3-4B (Yang et al., 2025), Llama-3.2-3B-Instruct (Grattafiori et al., 2024), and Phi-4-mini-Instruct (Abouelenin et al., 2025). Generation hyper-parameters and task-specific rewards are identical across baselines (Appendices B.2 and C).

4.2 Main Results (RQ1)

Tables 1 and 2 compare CERA-MoA with orchestration optimization methods and fixed-workflow fine-tuning approaches for both in-distribution (ID) and out-of-distribution (OOD) tasks. In the shared-backbone rows, four agents with independent LoRA adapters share Qwen3-4B, Llama-3.2-3B, or Phi-4-mini, respectively. CERA-MoA achieves the highest average performance across all three base models, demonstrating the broad applicability of our co-evolutionary framework across different model architectures and capacities.

Table 3: Ablation results on Qwen3-4B.

Method / Configuration	ID Avg.	OOD Avg.	Avg. Tokens
CERA-MoA (Adaptive Threshold)	63.2	72.8	367.77
Routing Metric (RQ3)			
Direct Reward Regression	60.8	70.1	
Multi-Class Classification	61.3	70.4	
Allocation Strategy (RQ4)			
CERA-MoA (Fixed Top-1)	61.4	70.1	325.70
CERA-MoA (Fixed Top-2)	63.3	72.5	666.37

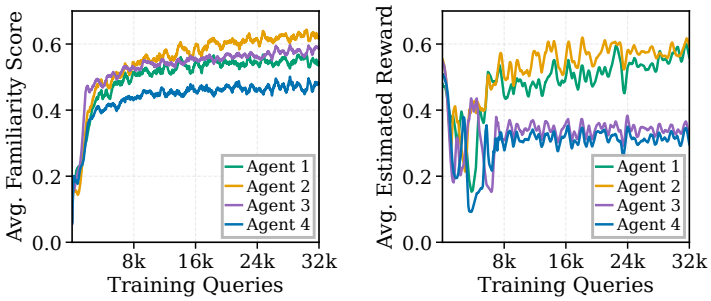


Figure 2: Comparison of routing metric dynamics.

For in-distribution tasks, our framework achieves substantial average gains. This improvement highlights the benefit of dynamically allocating training queries based on evolving agent capabilities, which enables more targeted policy updates and complementary specialization. For out-of-distribution benchmarks, CERA-MoA still achieves the highest average transfer scores. These results suggest that the learned capability differentiation can transfer beyond the training distribution, allowing the agents to leverage their learned competencies on unseen tasks.

4.3 Adaptation to Heterogeneous Models (RQ2)

We evaluate CERA-MoA on a heterogeneous ensemble combining Qwen3-4B, Llama-3.2-3B, and Phi-4-mini. As shown in the bottom rows of Tables 1 and 2, CERA-MoA still outperforms all baselines, achieving the highest average scores for both ID tasks and OOD transfer.

Fixed-workflow paradigms typically feed all training data to every agent without distinction, which ignores differences in pretrained strengths. Routing by familiarity instead sends each query to the agents that currently match it. The gap is largest on hard reasoning sets (DAPO-MATH, LogicBench, and OlympiadBench), where model diversity becomes a collaborative advantage. These findings suggest that CERA-MoA exhibits strong adaptability across different model architectures, effectively turning model diversity into a collaborative advantage.

4.4 Ablation on Familiarity Score (RQ3)

We compare the predictive familiarity estimator against two alternative routing mechanisms based on the same hidden states. As reported in Table 3, direct reward regression and reward-driven multi-class classification both fall short of CERA-MoA.

Figure 2 plots familiarity scores and estimated rewards across training queries. While estimated rewards exhibit oscillations, our familiarity score evolves smoothly and converges steadily. Direct reward regression and reward-driven optimization fail because rewards are inherently unstable. As agent policies evolve during training, their performance on identical prompts constantly changes. In contrast, our predictive familiarity estimator translates these dynamic shifts into a geometric push-and-pull distance between two network heads. This mechanism naturally adapts to continual learning dynamics, ensuring stable sample allocation and deeper specialization. Appendix E.5 further shows that mid-layer features outperform final-layer features.

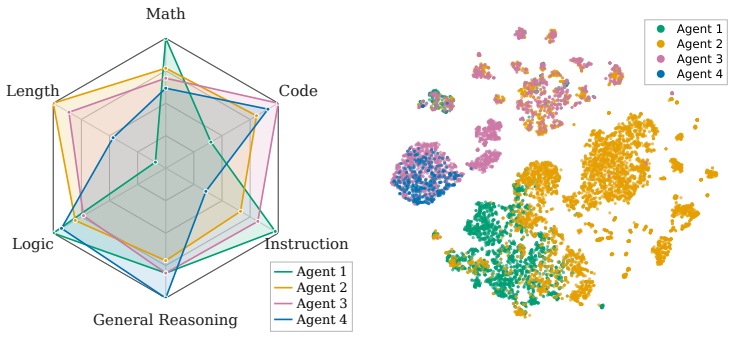


Figure 3: Capability analysis of Qwen3-4B agents.

4.5 Ablation on Adaptive Routing (RQ4)

We evaluate how our cumulative-threshold adaptive routing strategy balances reasoning accuracy against computational overhead. Table 3 compares our dynamic allocation mechanism against fixed Top-k activation strategies on Qwen3-4B.

Within our framework, enforcing a fixed Top-2 routing strategy yields strong reasoning accuracy, but incurs high token costs. Conversely, a fixed Top-1 strategy reduces computational overhead, however, it suffers a notable performance drop because a single agent struggles to resolve complex queries alone. Our cumulative-threshold mechanism resolves this trade-off. The router generally activates multiple agents only for complex problems that exceed an individual agent's expertise. Compared with fixed Top-2 routing, CERA-MoA reduces average generation tokens by approximately 45% while retaining comparable performance.

4.6 Capability Specialization Analysis (RQ5)

We investigate how our closed-loop co-evolution explicitly promotes distinct problem-solving specialization without human intervention or role assignment. We analyze the emergent behavioral profiles of the four Qwen3-4B agents.

As shown in Figure 3 (Left), the agent population spontaneously differentiated into complementary specialists. Due to long chain-of-thought reasoning, Agent 2 was allocated the majority of queries as a primary solver. To supplement this foundational reasoning capacity, Agent 1 mastered concise math and logic, Agent 3 focused on coding, and Agent 4 specialized in general reasoning. Figure 3 (Right) explains this emergence through a t-SNE visualization of intermediate hidden states. Queries routed to the same agent form tightly clustered semantic neighborhoods. By consistently allocating similar prompts to the most receptive model during training, our predictive familiarity estimator drives deep policy differentiation.

5 Conclusion

This paper introduces CERA-MoA, which co-evolves query routing with continually learning LLM agents. By evaluating agent competence via familiarity scores based on hidden-state semantics and applying cumulative-threshold routing, CERA-MoA balances efficiency with reasoning accuracy while inducing domain specialization, outperforming static routing and fixed-workflow fine-tuning baselines.

For future work, extending our single-turn semantic routing to multi-turn interactions or multi-agent debate could better support iterative problem-solving. Additionally, upgrading our familiarity-weighted voting to advanced answer aggregation mechanisms, such as an adaptive meta-thinker, promises to further enhance collective reasoning.

AI USE STATEMENT

We used generative AI tools to revise wording for clarity and consistency of terminology. We did not use generative AI tools to generate experimental results, figures, or numerical claims. All AI-assisted text was reviewed by the authors, who take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Algorithm 1 and Appendix A give the full co-evolution procedure, head architecture, and inference rule. Hardware, software versions, hyper-parameters, and prompt templates are in Appendix B; reward functions are in Appendix C; data sources, per-source sample counts, and splits are in Appendix D. Appendix E reports router latency, training dynamics, sensitivity to τ, λ, and m, and ablations of exploration and hidden-state depth. All results use seed 42 and one training run per configuration. The 32k training sample is specified by per-source counts and that seed rather than per-example hashes; Appendix D discusses possible train–test prompt overlap in public source pools.

REFERENCES

Abdelrahman Abouelenin, Atabak Ashfaq, Adam Atkinson, Hany Awadalla, Nguyen Bach, Jianmin Bao, et al. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-loras. *arXiv preprint arXiv:2503.01743*, 2025.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation, 2018.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate. In *ICLR*, volume 2024, pp. 9079–9093, 2024.

Justin Chen, Swarnadeep Saha, and Mohit Bansal. Reconcile: Round-table conference improves reasoning via consensus among diverse llms. In *Proc. of ACL*, pp. 7066–7085, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, et al. Evaluating large language models trained on code, 2021.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018.

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

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, et al. Multi-agent collaboration via evolving orchestration. *NeurIPS*, 38: 165025–165059, 2026.

Andrew Estornell and Yang Liu. Multi-llm debate: Framework, principals, and interventions. *NeurIPS*, 37:28938–28964, 2024.

Wei Fan, JinYi Yoon, and Bo Ji. imad: Intelligent multi-agent debate for efficient and accurate llm inference. In *Proc. of AAAI*, volume 40, pp. 29403–29411, 2026.

Tao Feng, Haozhen Zhang, Zijie Lei, Peixuan Han, and Jiaxuan You. Graphplanner: Graph memory-augmented agentic routing for multi-agent llms. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 123309–123340, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

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

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *ICLR*, volume 2024, pp. 23247–23275, 2024.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

Tianyu Hu, Zhen Tan, Song Wang, Huaizhi Qu, and Tianlong Chen. Multi-agent debate for llm judges with adaptive stability detection. *NeurIPS*, 38:46504–46540, 2026.

Jiatan Huang, Zheyuan Zhang, Kaiwen Shi, Yanfang Ye, and Chuxu Zhang. Evolverouter: Co-evolving routing and prompt for multi-agent question answering, 2026.

Zixuan Ke, Yifei Ming, Austin Xu, Ryan Chin, Xuan-Phi Nguyen, Prathyusha Jwalapuram, et al. Mas-orchestra: Understanding and improving multi-agent reasoning through holistic orchestration and controlled benchmarks, 2026.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Hyunseok Lee, Seunghyuk Oh, Jaehyung Kim, Jinwoo Shin, and Jihoon Tack. ReVISE: Learning to refine at test-time via intrinsic self-verification. In *Proc. of ICML*, pp. 33616–33634. PMLR, 13–19 Jul 2025.

Sangmook Lee, Dohyung Kim, Hyukhun Koh, Nakyeong Yang, and Kyomin Jung. Confidence-guided stepwise model routing for cost-efficient reasoning. In *Proc. of AAAI*, volume 40, pp. 31483–31491, 2026.

Peiwen Li, Shiyang Zhang, Yangtian Zhang, Sizhuang He, David van Dijk, and Rex Ying. Morse: Task-oriented multi-agent system with mixture of role-subtask experts, 2026a.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.

Shiyuan Li, Yixin Liu, Yu Zheng, Mei Li, Quoc Viet Hung Nguyen, and Shirui Pan. Ofa-mas: One-for-all multi-agent system topology design based on mixture-of-experts graph generative models. In *Proc. of WWW*, pp. 1333–1344, 2026b.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *Proc. of EMNLP*, pp. 17889–17904, 2024.

Ang Lv, Jin Ma, Yiyuan Ma, and Siyuan Qiao. Coupling experts and routers in mixture-of-experts via an auxiliary loss, 2025.

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip H. S. Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. Malt: Improving reasoning with multi-agent llm training, 2025.

James Oldfield, Markos Georgopoulos, Grigorios G Chrysos, Christos Tzelepis, Yannis Panagakis, Mihalis A Nicolaou, Jiankang Deng, and Ioannis Patras. Multilinear mixture of experts: Scalable expert specialization through factorization. *NeurIPS*, 37:53022–53063, 2024.

Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman E Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. Maporl: Multi-agent post-co-training for collaborative large language models with reinforcement learning. In *Proc. of ACL*, pp. 30215–30248, 2025.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, et al. Logicbench: Towards systematic evaluation of logical reasoning ability of large language models, 2024.

Manhin Poon, XiangXiang Dai, Xutong Liu, Fang Kong, John CS Lui, and Jinhang Zuo. Online multi-llm selection via contextual bandits under unstructured context evolution. In *Proc. of AAAI*, volume 40, pp. 24855–24863, 2026.

Haritz Puerto, Tilek Chubakov, Xiaodan Zhu, Harish Tayyar Madabushi, and Iryna Gurevych. Fine-tuning on diverse reasoning chains drives within-inference cot refinement in llms. In *Proc. of ACL*, pp. 3789–3808, 2025.

Dan Qiao, Binbin Chen, Fengyu Cai, Jianlong Chen, Wenhao Li, Fuxin Jiang, Zuzhi Chen, Hongyuan Zha, Tieying Zhang, and Baoxiang Wang. Epistemic gain, aleatoric cost: Uncertainty decomposition in multi-agent debate for math reasoning, 2026.

Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Nikul Patel, Jalal Naghiyev, Yann Lecun, and Ravid Shwartz-Ziv. Layer by layer: Uncovering hidden representations in language models. In *Proc. of ICML*, pp. 55854–55875. PMLR, 2025.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed H Chi, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of ACL*, pp. 13003–13051, 2023.

Jinzhou Tang, Jusheng Zhang, Qinhan Lv, Sidi Liu, Jing Yang, Chengpei Tang, and Keze Wang. Hiva: Self-organized hierarchical variable agent via goal-driven semantic-topological evolution, 2025.

Amir Taubenfeld, Tom Sheffer, Eran Ofek, Amir Feder, Ariel Goldstein, Zorik Gekhman, and Gal Yona. Confidence improves self-consistency in llms. In *Findings of ACL*, pp. 20090–20111, 2025.

Chenxu Wang, Hao Li, Yiqun Zhang, Linyao Chen, Jianhao Chen, Ping Jian, Qiaosheng Zhang, and Shuyue Hu. Icl-router: In-context learned model representations for llm routing. In *Proc. of AAAI*, volume 40, pp. 33413–33421, 2026a.

Hongyi Wang, Felipe Maia Polo, Yuekai Sun, Souvik Kundu, Eric Xing, and Mikhail Yurochkin. Fusing models with complementary expertise. In *ICLR*, volume 2024, pp. 45284–45306, 2024.

Jingbo Wang, Sendong Zhao, Jiatong Liu, Haochun Wang, Wanting Li, Bing Qin, and Ting Liu. Orchestrating intelligence: Confidence-aware routing for efficient multi-agent collaboration across multi-scale models. *arXiv preprint arXiv:2601.04861*, 2026b.

Jize Wang, Han Wu, Zhiyuan You, Yiming Song, Yijun Wang, Zifei Shan, Yining Li, Songyang Zhang, Xinyi Le, Cailian Chen, et al. Routemoa: Dynamic routing without pre-inference boosts efficient mixture-of-agents. *arXiv preprint arXiv:2601.18130*, 2026c.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Y Zou. Mixture-of-agents enhances large language model capabilities. In *ICLR*, volume 2025, pp. 33944–33963, 2025a.

Xinyuan Wang, Yanchi Liu, Wei Cheng, Xujiang Zhao, Zhengzhang Chen, Wenchao Yu, et al. Mixllm: Dynamic routing in mixed large language models. In *Proc. of NAACL*, pp. 10912–10922, 2025b.

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 *Proc. of ACL*, pp. 24013–24035, 2025c.

Zhexuan Wang, Xuebo Liu, Li Wang, Zifei Shan, Yutong Wang, Zhenxi Song, and Min Zhang. Maspo: Joint prompt optimization for llm-based multi-agent systems, 2026d.

Xingjian Wu, Junkai Lu, Siyu Yan, Xiangfei Qiu, Jilin Hu, Chenjuan Guo, and Bin Yang. Differentiable mixture-of-agents incentivizes swarm intelligence of large language models, 2026.

Yu Xia, Fang Kong, Tong Yu, Liya Guo, Ryan A Rossi, Sungchul Kim, and Shuai Li. Which llm to play? convergence-aware online model selection with time-increasing bandits. In *Proc. of WWW*, pp. 4059–4070, 2024.

Zhentao Xie, Chengcheng Han, Jinxin Shi, Wenjun Cui, Wayne Xin Zhao, Xingjiao Wu, and Jiabao Zhao. Rmoa: Optimizing mixture-of-agents through diversity maximization and residual compensation. In *Findings of ACL*, pp. 6575–6602, 2025.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. In *ICLR*, volume 2025, pp. 76346–76382, 2025.

Jiaqi Xue, Qian Lou, Jiarong Xing, and Heng Huang. R2-router: A new paradigm for llm routing with reasoning, 2026a.

Xiangyuan Xue, Yifan Zhou, Guibin Zhang, Zaibin Zhang, Yijiang Li, Chen Zhang, Zhenfei Yin, Philip Torr, Wanli Ouyang, and Lei Bai. Comas: Co-evolving multi-agent systems via interaction rewards, 2026b.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. Mas-gpt: Training llms to build llm-based multi-agent systems. *arXiv preprint arXiv:2503.03686*, 2025.

Xie Yi, Zhanke Zhou, Chentao Cao, Qiyu Niu, Tongliang Liu, and Bo Han. From debate to equilibrium: Belief-driven multi-agent llm reasoning via bayesian nash equilibrium. *arXiv preprint arXiv:2506.08292*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *NeurIPS*, 38:113222–113244, 2026.

Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. Free process rewards without process labels. *arXiv preprint arXiv:2412.01981*, 2024.

Yurun Yuan and Tengyang Xie. Reinforce llm reasoning through multi-agent reflection. In *ICML*, pp. 73701–73731. PMLR, 2025.

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 *Proc. of ACL*, pp. 15549–15572, 2025.

Guibin Zhang, Haiyang Yu, Kaiming Yang, Bingli Wu, Fei Huang, Yongbin Li, and Shuicheng Yan. Evoroute: Experience-driven self-routing llm agent systems, 2026.

Kaiyan Zhang, R Liu, X Zhu, K Tian, S Zeng, Guoli Jia, Yuchen Fan, Xingtai Lv, Yuxin Zuo, Che Jiang, et al. Marti: A framework for multi-agent llm systems reinforced training and inference. In *The Fourteenth ICLR*, 2025.

Xiutian Zhao, Ke Wang, and Wei Peng. An electoral approach to diversify llm-based multi-agent collective decision-making. In *Proc. of EMNLP*, pp. 2712–2727, 2024.

Xuandong Zhao, Zhewei Kang, Aosong Feng, Sergey Levine, and Dawn Song. Learning to reason without external rewards, 2025.

Yujie Zhao, Lanxiang Hu, Yang Wang, Minmin Hou, Hao Zhang, Ke Ding, and Jishen Zhao. Stronger-mas: Multi-agent reinforcement learning for collaborative llms, 2026.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization, 2025.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, et al. Agieval: A human-centric benchmark for evaluating foundation models, 2023.

Heng Zhou, Hejia Geng, Xiangyuan Xue, Li Kang, Yiran Qin, Zhiyong Wang, Zhenfei Yin, and Lei Bai. ReSo: A reward-driven self-organizing LLM-based multi-agent system for reasoning tasks. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 15979–15998, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.808.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, et al. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Junze Zhu, Weihao Chen, Xuanwang Zhang, Zhen Wu, and Xinyu Dai. Recognize your orchestrator: An entropy dynamics perspective for llm multi-agent systems, 2026.

Yuan Zhuang, Yi Shen, Yuexin Bian, Qing Su, Shihao Ji, Yuanyuan Shi, and Fei Miao. Ld-mole: Learnable dynamic routing for mixture of lora experts. *arXiv preprint arXiv:2509.25684*, 2025.

A Algorithm and Implementation Details

A.1 Co-Evolutionary Training Procedure

Algorithm 1 CERA-MoA Co-Evolutionary Training

Require: Training set $\mathcal{D}$; agent policies $\{\pi_{\theta_i}\}_{i=1}^{N}$; trainable predictor heads $\{g_i\}_{i=1}^{N}$; frozen target heads $\{\bar{g}_i\}_{i=1}^{N}$; threshold τ; margin m

Initialize an empty LIFO buffer $\mathcal{B}_i$ for each agent $i \in \{1, \dots, N\}$
for each orchestration step **do**
  Sample a query batch $\mathcal{Q} \subset \mathcal{D}$ and compute frozen features $h(q)$ for all $q \in \mathcal{Q}$ (Eq. 2)
  for each query $q \in \mathcal{Q}$ **do**
    Compute familiarity scores $f_i(q)$ (Eq. 4) and routing scores $s_i(q)$ (Eq. 5)
    Rank agents by $s_i(q)$ in descending order
    Activate the shortest prefix $\mathcal{S}_q$ of this ranking with $\sum_{i \in \mathcal{S}_q} f_i(q) \geq \tau$ (Eq. 6); if no prefix qualifies, set $\mathcal{S}_q = \{1, \dots, N\}$
    Push $(q, h(q))$ into $\mathcal{B}_i$ for every $i \in \mathcal{S}_q$
  end for
  for each agent i whose buffer $\mathcal{B}_i$ holds a full micro-batch **do**
    Pop the most recent micro-batch Q_i from $\mathcal{B}_i$
    Sample G completions $\{a_{i,q}^{(g)}\}_{g=1}^{G}$ for each $q \in Q_i$ and score them with the task reward R
    Update θ_i by minimizing $\mathcal{L}_{\text{agent}}(\theta_i)$ (Eq. 12)
    Record the mean reward $r_{i,q} = \frac{1}{G} \sum_{g=1}^{G} R(q, a_{i,q}^{(g)})$ for each $q \in Q_i$
  end for
  for each routed query q whose rewards $\{r_{j,q}\}_{j \in \mathcal{S}_q}$ have been recorded, and each $i \in \mathcal{S}_q$ **do**
    Compute $A_{i,q} = r_{i,q} - \frac{1}{|\mathcal{S}_q|} \sum_{j \in \mathcal{S}_q} r_{j,q}$, or $A_{i,q} = r_{i,q}$ if $|\mathcal{S}_q| = 1$
    Update g_i by minimizing $\mathcal{L}_{\text{router}}$ (Eq. 15), keeping $\bar{g}_i$ and the encoder frozen
  end for
end for

Algorithm 1 summarizes training. Each orchestration step routes a batch of queries, updates every agent whose buffer is full, and then updates the predictor heads of the agents that were evaluated. The procedure does not depend on how agents are parameterized. In the shared-backbone setting,

each agent is a LoRA adapter on one frozen base model, so all agents share the base weights in GPU memory. In the heterogeneous setting, each agent is a different base model with its own adapter. In both settings the encoder that produces $h(q)$ is frozen, and the only trainable parameters are the adapters θ_i and the predictor heads g_i.

Buffers and update frequency. An agent is updated only when its buffer holds a full micro-batch ($|Q_i| = 16$). Agents that are activated more often are therefore also updated more often, so the router controls both which queries each agent learns from and how frequently it learns (Figure 5b). The LIFO order makes each update use the most recently routed queries, i.e., those assigned under the latest familiarity estimates.

Predictor and target heads. All $2N$ heads share one architecture,

$$\mathrm{LN}(h) \to \mathrm{Linear}(2H, 512) \to \mathrm{SiLU} \to \mathrm{Linear}(512, 256), \tag{16}$$

where H is the hidden size of the encoder and $h(q) \in \mathbb{R}^{2H}$ is the concatenation in Eq. 2. Each head has roughly $10^3 H$ parameters, orders of magnitude fewer than an agent policy. Linear layers are orthogonally initialized (Section 3.2), and head outputs are ℓ_2-normalized before computing $d_i(q)$ (Eq. 3). At initialization, the outputs of a predictor and its target are close to orthogonal in $\mathbb{R}^{256}$, so $d_i(q) \approx \sqrt{2}$ and $f_i(q) \approx e^{-2\sqrt{2}} \approx 0.06$ for $\lambda = 2$. The four scores then sum to about $0.24 < \tau$, so early training queries fall back to the full pool (Figure 9b), and allocation narrows only as predictor heads are pulled toward their targets.

Range of familiarity and the margin condition. With $\lambda = 2$, $f_i(q) \in [e^{-4}, 1] \approx [0.018, 1]$, and a single agent meets $\tau = 0.7$ on its own only if $d_i(q) \le \lambda^{-1} \log(1/\tau) \approx 0.18$. The negative branch of Eq. 15 stops pushing once $d_i(q) \ge m$, so an agent that keeps underperforming on a region of queries settles near $f_i = e^{-\lambda m} \approx 0.14$ rather than at the floor $e^{-2\lambda}$. We choose m so that $Ne^{-\lambda m} \le \tau$: with $m = 1$, this gives $4e^{-2} \approx 0.54$ for $N = 4$ and $3e^{-2} \approx 0.41$ for $N = 3$, both below $\tau = 0.7$. A query on which every agent has been pushed to the margin therefore cannot meet the threshold and is sent to the whole pool, which gives every agent another chance to earn a positive advantage on it.

Exploration-augmented ranking. Eq. 5 decides the order in which agents are considered, whereas Eq. 6 decides how many are activated, using f_i only. Here T counts routed queries, n_i counts queries allocated to agent i, and $\bar{H}_i$ is the running mean generation entropy of agent i's completions. Keeping the bonuses out of the cutoff matters: with $c_{\mathrm{ucb}} = 2$, the UCB term alone exceeds $\tau = 0.7$ whenever $\log T/(n_i + 1) > 0.12$, so a cutoff on s_i would activate a single under-used agent regardless of its competence. The bonuses can move an agent forward in the ranking, but the number of activated agents still reflects familiarity alone.

Policy optimization. Each agent minimizes Eq. 12 on its own micro-batch. Advantages are normalized within the G completions that one agent produced for one query, so agents never compete in the policy loss; cross-agent comparison enters only through the router. If all G completions receive the same reward, their advantages are zero and the query contributes only through the KL term. The importance ratio is computed per sequence as in GSPO and clipped with $\epsilon = 0.2$. Following DAPO, the loss is averaged over all completion tokens in the batch rather than per sequence, and overlong completions are penalized softly through the reward (Appendix C). The k3 estimator is non-negative and, for on-policy samples, unbiased for $\mathrm{KL}(\pi_{\theta_i} \,\|\, \pi_{\mathrm{ref}})$.

Router optimization. The router uses the cross-agent advantage $A_{i,q}$, which compares agent i with the other agents activated on the same query, unlike the within-agent advantage $\hat{A}^{(g)}_{i,q}$ used for policy updates. A positive $A_{i,q}$ pulls $g_i(h(q))$ toward $\bar{g}_i(h(q))$, which raises f_i on q and, through the shared features $h(\cdot)$, on similar queries. A negative $A_{i,q}$ pushes the predictor away until $d_i(q) = m$. When a single agent is activated, there is no within-subset baseline and we use its mean reward: a successful solo agent is pulled closer, an agent with zero reward is left unchanged, and one with negative reward (possible through the length penalty) is pushed away. The margin bounds the repulsive branch. Without it, repeated negative advantages would keep pushing a head until f_i reaches its floor $e^{-2\lambda}$, after which many positive updates are needed to recover; Appendix E.3 shows that removing the margin destabilizes training.

A.2 Inference and Aggregation

At test time we set $c_{\mathrm{ucb}} = w_{\mathrm{ent}} = 0$, so agents are ranked by $f_i(q)$ and the shortest prefix whose cumulative familiarity reaches τ is activated, with the full pool as fallback. Each activated agent generates one completion. For tasks with a deterministic extracted answer, completions are grouped by $\mathrm{ans}(\cdot)$ (boxed answer, choice letter, or constraint-verifier output; Appendix C), each group is scored by the sum of its members' familiarity, and CERA-MoA returns the completion of the most familiar agent in the highest-scoring group. For code generation, where exact agreement between programs is rare, CERA-MoA returns the completion of the most familiar activated agent.

B Experimental Setup Details

B.1 Hardware and Software

All experiments run on one node with four NVIDIA H200 GPUs (141 GB HBM3e each), two Intel Xeon Platinum 8558 CPUs (96 physical cores in total), and about 500 GB of host memory, under Ubuntu 22.04.5 LTS. Training and inference use PyTorch with `bfloat16` mixed precision; Table 4 lists library versions. The TRL components come from a repository-local modification that adds group-normalized DAPO losses and GSPO sequence-level importance ratios.

Table 4: Main software libraries used for training, reward evaluation, and data loading.

Library	Version
Python	3.12
PyTorch	2.9.1
Transformers	4.57.6
PEFT	0.11.1
vLLM	0.16.0
Accelerate	1.13.0
Datasets	4.8.4
Safetensors	0.7.0
SymPy	1.14.0
NumPy	2.2.6
Pandas	2.2.2

Rollouts are generated by vLLM engines co-located with training, and re-entrant gradient checkpointing in the policy backward pass keeps activation memory within budget. In the shared-backbone setting, the four LoRA adapters are served on a single copy of the base model. In the heterogeneous setting, each base model and its adapter run in a dedicated worker process, which avoids CUDA-context contention and memory fragmentation between the training stack and the vLLM engine. One thousand training steps (16,000 training queries with $G = 8$ rollouts each) take about one H200 GPU-day.

B.2 Hyper-Parameters

Table 5 lists all hyper-parameters; baselines use the same generation settings and rewards. Each orchestration step samples $B_{\mathrm{global}} = B_{\mathrm{device}} \times A/G = 16 \times 8/8 = 16$ prompts, where B_{device} is the per-device batch size, A the gradient accumulation factor, and G the number of rollouts per prompt. Because an agent is updated only when its buffer holds $|Q_i| = 16$ queries, the per-agent update frequency follows the router's allocation rather than a fixed epoch schedule: agents that become familiar with a region of queries receive more of them and are updated more often, without a manual curriculum.

B.3 Prompt Templates

Agents in CERA-MoA receive only a task-family template around the raw query (Figure 4); no system prompt, persona, or role description is added. The specialization in Section 4.6 therefore arises from routing and reinforcement learning rather than from prompts. Orchestration baselines, whose

Table 5: Hyper-parameters for CERA-MoA training and inference.

Hyper-Parameter	**Value**
Random seed	42
Agent pool size N	4 (shared backbone); 3 (heterogeneous)
LoRA	Rank $r = 16$, $\alpha = 32$, dropout 0.05
LoRA target modules	`q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj`
Learning rates	Predictor heads 1×10^{-4}; agent adapters 3×10^{-5}
Agent learning-rate schedule	Cosine decay, 600 warm-up steps, floor 2×10^{-5}
Routing	Distance decay $\lambda = 2.0$; threshold $\tau = 0.7$; margin $m = 1.0$
Exploration	$c_{\text{ucb}} = 2.0$; $w_{\text{ent}} = 2.0$ (shared backbone) / 0.2 (heterogeneous)
Policy objective	DAPO loss with group-normalized rewards and GSPO sequence-level ratios
Rollouts per prompt G	8
Regularization	KL weight $\beta = 1 \times 10^{-4}$; clip range $\epsilon = 0.2$
Batching	Per-device batch 16 prompts; gradient accumulation $A = 8$
Training length	At most 6,000 optimizer updates; 2 trainer steps per rollout generation
Optimizer	AdamW, $\beta_1 = 0.9$, $\beta_2 = 0.95$; agent weight decay 1×10^{-3}
Gradient clipping	Norm 1.0 for both predictor heads and agents
Generation	Temperature 1.0; top-p 0.95; top-k 50; repetition penalty 1.0
Sequence lengths	Prompt $\leq$ 800 tokens; completion $\leq$ 1,000 tokens ($L_{\max}$)

agents would otherwise be identical, use role-specific prompts to simulate four experts (Section 4.1). Built-in thinking modes (e.g., that of Qwen3) are disabled for all models so that generation lengths are comparable across base models, and prompts longer than 800 tokens are truncated from the left before routing and generation.

C Task-Specific Rewards

Each completion receives a task reward $r_{\text{base}} \in [0, 1]$ minus a soft length penalty. Let ℓ be the number of generated tokens, $L_{\max}$ the maximum completion length, $\gamma \in (0, 1)$ the fraction of $L_{\max}$ at which the penalty starts, and $\eta \geq 0$ the maximum penalty. The training reward is

$$R = r_{\text{base}} - \eta \cdot \begin{cases} 0, & \ell \leq \gamma L_{\max}, \\ \dfrac{\ell - \gamma L_{\max}}{(1-\gamma) L_{\max}}, & \gamma L_{\max} < \ell < L_{\max}, \\ 1, & \ell \geq L_{\max}. \end{cases} \tag{17}$$

The penalty is zero up to $\gamma L_{\max}$ and grows linearly to η at the completion limit. For evaluation, a response counts as correct only if $r_{\text{base}} = 1$; for code and instruction following this means that all tests or all constraints pass, so the fractional rewards below serve only as training signals.

Mathematical reasoning. The evaluator finds all occurrences of `\boxed{}` and takes the last complete balanced-brace expression, so that intermediate derivations are not treated as the final answer. It then normalizes common presentation variants, including surrounding delimiters and whitespace, units, thousands separators, reducible fractions, percentage markers, matrix delimiters, and multiple-choice notation, and additionally handles set-style answers, $\pm$ expansions, ratios, and LaTeX expressions. Prediction and reference are compared by exact matching followed by SymPy simplification and equivalent algebraic transformations, under a timeout so that malformed expressions cannot block training:

$$r_{\text{base}}^{\text{math}} = \mathbb{I}\{\text{MathEqual}(\text{Extract}(a), y^\star)\}.$$

This accepts mathematically equivalent answers and falls back to exact matching when symbolic parsing fails.

Code generation. The evaluator extracts the last fenced Python block, or the raw completion if there is none. Code runs in an isolated subprocess under a reliability guard, with a 10-second wall-

Task-Specific Zero-Shot Prompt Templates across Benchmark Domains

[Domain: Mathematical Reasoning]

Problem:
{question}
Please reason step by step.
Present the final answer in the following format:
Answer: \boxed{XX}

[Domain: General Reasoning]

Problem:
{question}
Please reason step by step.
Present the final answer in the following format:
Answer: \boxed{XX}

[Domain: Code Generation for a function]

Problem:
{question}
Please reason step by step.
Write a Python function named {fn_name} to solve the problem.
Present the code in:
```python
Your code
```

[Domain: Code Generation]

Problem:
{question}
Please reason step by step.
Write Python code to solve the problem.
Present the code in:
```python
Your code
```

[Domain: Open-Ended Instruction Following]

{question}

Figure 4: Zero-shot prompt templates for each task family. The two code-generation templates correspond to function-call and standard-input/output tasks. {question} and {fn_name} are instantiated from each dataset item. Instruction-following prompts are passed through unchanged so that their verifier-defined constraints are preserved.

clock limit covering compilation and all tests; both standard-input/output programs and function-call tasks are supported. The training reward is the fraction of passed tests,

$$r_{\text{base}}^{\text{code}} = \frac{1}{|\mathcal{T}|} \sum_{(x,y)\in\mathcal{T}} \mathbb{I}\{\text{Run}(a,x) = y\}.$$

Compilation errors, runtime errors, timeouts, malformed outputs, and missing code receive zero reward.

Instruction following. Each instruction-following item stores a list of instruction identifiers and their argument dictionaries, from which the official IFEval (Zhou et al., 2023) registry reconstructs the verifiers. The training reward is the fraction of satisfied constraints,

$$r_{\text{base}}^{\text{IF}} = \frac{1}{K} \sum_{k=1}^{K} \mathbb{I}\{\text{Check}_k(a; q, \omega_k) = 1\}.$$

Constraints include exact endings, required keywords, case restrictions, sentence or bullet counts, and markup requirements. Empty responses and unavailable registry entries receive zero reward.

General reasoning. General-reasoning items use the same final-answer extractor as math, followed by task-agnostic normalization: optional "Answer:" prefixes and enclosing parentheses are

removed, single-letter choices are uppercased, and the remaining text is whitespace-canonicalized and lowercased. The reward is exact match after normalization,

$$r_{\text{base}}^{\text{general}} = \mathbb{I}\{\text{Normalize}(\text{Extract}(a)) = \text{Normalize}(y^\star)\}.$$

D DATASETS

Table 6: Data sources, training pools, the number of examples sampled into the 32k training set, and test-set sizes. OOD benchmarks are used only for evaluation.

Domain	Dataset	Train pool	Sampled	Test
Math	GSM8K (Cobbe et al., 2021)	7,473	1,200	1,319
	MATH (Hendrycks et al., 2021)	7,500	5,000	5,000
	DAPO-MATH-17k (Yu et al., 2026)	13,616	5,000	500
Code	MBPP, sanitized (Austin et al., 2021)	163	163	257
	Eurus-2-Code (Yuan et al., 2024)	25,278	7,000	1,024
	TACO (Li et al., 2023)	24,702	2,437	1,000
Instr.	MAGPIE-IF (Xu et al., 2025)	551,465	1,400	500
	RLVR-IFEval (Lambert et al., 2024)	14,540	5,000	500
General	BBH (Suzgun et al., 2023)	6,011	4,800	500
Total (ID)			**32,000**	
OOD	IFEval (Zhou et al., 2023)	–	–	541
	HumanEval (Chen et al., 2021)	–	–	154
	AGIEval (Zhong et al., 2023)	–	–	1,000
	ARC-Challenge (Clark et al., 2018)	–	–	1,172
	LogicBench (Parmar et al., 2024)	–	–	1,520
	OlympiadBench (He et al., 2024)	–	–	707

Table 6 lists every source, the size of its training pool, the number of examples drawn into our training set, and the size of its test split.

In-distribution sources. GSM8K contains grade-school word problems that require multi-step arithmetic and have numeric answers. MATH contains competition problems in algebra, geometry, number theory, counting, and probability, with answers in mathematical notation that require equivalence-aware checking. DAPO-MATH-17k is a verifiable mathematical reasoning collection; we use a processed version with a 500-item test split. MBPP (sanitized) consists of function-level Python tasks with executable input–output examples. Eurus-2-Code and TACO contain programming problems with executable tests; TACO also includes standard-input/output problems that evaluate complete programs. MAGPIE-IF (the IFEval-like subset derived from MAGPIE) and RLVR-IFEval contain prompts whose constraints on content, structure, length, casing, and ordering can be checked programmatically; each item stores the instruction identifiers and arguments needed to rebuild its verifiers. BIG-bench Hard (BBH) covers challenging reasoning tasks, including symbolic manipulation, logical deduction, temporal and spatial reasoning, and language understanding.

Training set and ID evaluation. The 32,000 training examples are sampled from the training pools with a fixed seed (`seed = 42`) using the per-source counts in Table 6: 11,200 math, 9,600 code, 6,400 instruction-following, and 4,800 general-reasoning examples. All sources are converted into one JSON schema with fields `problem`, `class`, and task-specific verifier metadata: math and general-reasoning items provide `true_answer`; code items provide input–output `test_cases` and, where applicable, `fn_name`; instruction-following items provide `instruction_id_list` and the verifier arguments. ID results in Table 1 are computed separately on the held-out test split of each of the nine sources.

OOD evaluation. IFEval (541 prompts) measures compliance with verifiable instructions using deterministic checks. HumanEval (154 problems in our processed version) asks for Python functions from a signature and docstring and checks them with unit tests. AGIEval (1,000 questions)

contains multiple-choice questions from standardized examinations. ARC-Challenge (1,172 questions) targets grade-school science that requires combining facts. LogicBench (1,520 instances) tests formal and natural-language logical reasoning. OlympiadBench (707 problems) contains olympiad-level mathematics and science problems. None of these benchmarks contributes training data. OOD evaluation reuses the trained router, agents, prompt templates, and aggregation rule without any adaptation, so it measures how well the learned routing and specialization transfer.

Train–test overlap. The training sample is drawn only from the training pools, and ID results use the corresponding held-out splits. We did not run additional deduplication across sources. Some public pools are compiled from overlapping upstream collections, so a small number of training prompts may also appear, verbatim or nearly so, in an ID test split. Every method in Table 1 uses the same training sample and test splits, so any such overlap is shared across all rows; the OOD benchmarks in Table 2 are never used for training.

E Additional Experiments

Unless stated otherwise, the experiments in this section use the Qwen3-4B shared-backbone pool with $N = 4$ agents and the default hyper-parameters in Table 5. ID and OOD averages are computed as in Tables 1 and 2, and "Avg. Tokens" is the average number of generated tokens per test query, summed over activated agents.

E.1 Router Overhead

Table 7: Per-query latency of routing and generation on one NVIDIA H200 GPU.

Component	Batch Size	Avg. Latency (ms)
Familiarity router	1	11.87
Completion generation (native)	1	1350.32
Completion generation (vLLM)	32	154.09

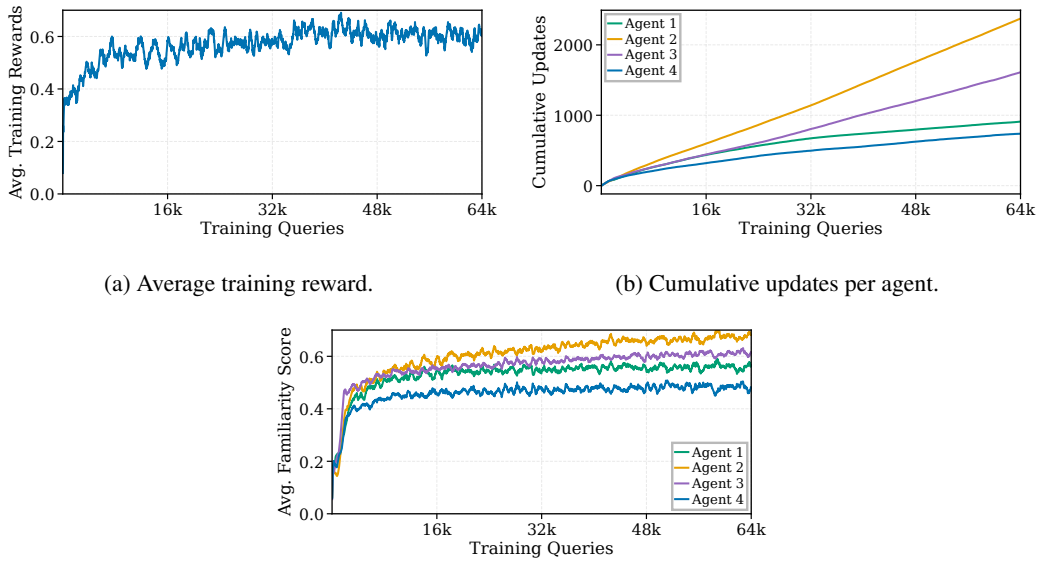
Table 7 compares the time needed to route a query with the time needed to generate one completion. Computing all N familiarity scores takes 11.87 ms per query, because the router needs only one forward pass of the frozen encoder up to layer $\lfloor 3L/4 \rfloor$ followed by $2N$ small MLP heads, with no decoding. This is under 1% of the time needed to generate one completion without batching (1350.32 ms) and about 8% of the amortized per-query time of batched vLLM generation (154.09 ms at batch size 32). The router was timed at batch size 1, so batching would lower its per-query cost further. By comparison, adaptive routing saves about 300 generated tokens per query relative to fixed Top-2 routing (Table 3), so the time spent on routing is small next to the generation it avoids.

E.2 Training Dynamics

Figure 5 tracks training of the default pool. The average training reward (Figure 5a) rises from about 0.35 to about 0.6 during the first 16k queries and then stays near that level. Update counts (Figure 5b) separate early and keep diverging: after 64k queries, Agent 2 has performed roughly 2,400 updates and Agent 3 roughly 1,600, while Agents 1 and 4 have performed fewer than 1,000 each. With 16 queries per update, these counts correspond to about 1.4 activated agents per training query. Familiarity scores (Figure 5c) rise quickly for all agents in the first few thousand queries and then settle in the same order as the update counts, with Agent 2 highest and Agent 4 lowest. This matches the roles in Section 4.6: Agent 2 acts as the primary solver, Agent 3 handles most code queries, and Agents 1 and 4 cover concise math, instruction following, and general reasoning. The less-used agents still gain familiarity on their own query regions instead of collapsing, which is the purpose of the exploration terms (Appendix E.4).

E.3 Hyper-Parameter Sensitivity

Cumulative threshold τ. We first vary τ at test time for the default pool trained with $\tau = 0.7$ (Table 8). Accuracy changes little: for $\tau \in [0.5, 0.9]$, the ID average stays between 62.9 and 63.2



(a) Average training reward.

(b) Cumulative updates per agent.

(c) Average familiarity score per agent.

Figure 5: Training dynamics of the default Qwen3-4B pool over 64k training queries.

and the OOD average between 71.9 and 72.9. Cost changes much more. Lowering τ to 0.5 saves about 10% of generated tokens, whereas raising it to 0.9 increases them by 76% without improving accuracy. The threshold can therefore be adjusted at deployment to meet a compute budget without retraining.

Table 8: Test-time sweep of τ for the default pool trained with $\tau = 0.7$.

Test Threshold (τ)	ID Avg.	OOD Avg.	Avg. Tokens
0.5	62.9	71.9	329.06
0.6	63.0	72.5	342.22
0.7 (default)	63.2	72.8	367.77
0.8	63.0	72.9	422.12
0.9	63.2	72.7	648.01

We then retrain the pool with $\tau = 0.8$ (Table 9). At every test threshold, this pool is about 2 ID points below the default pool and generates more tokens; its best averages are 61.2 ID and 71.5 OOD. A stricter training threshold activates more agents per training query, so each agent receives a broader mix of queries and each relative advantage is computed against a larger subset. We attribute the drop to this weaker specialization signal. The training threshold thus matters more than the test threshold, and we use $\tau = 0.7$ in all other experiments.

Table 9: Test-time sweep of τ for a pool trained with $\tau = 0.8$.

Test Threshold (τ)	ID Avg.	OOD Avg.	Avg. Tokens
0.5	60.9	70.9	353.84
0.6	61.2	71.5	371.32
0.7	60.8	71.4	405.36
0.8 (training value)	61.2	71.4	485.88
0.9	61.2	71.5	676.93

Distance decay λ. Reducing λ from 2.0 to 1.0 lowers accuracy to 55.3 ID and 59.8 OOD (from 63.2 and 72.8) and raises the token count to 463.72. Figure 6 shows why. With $\lambda = 1$, every

agent's average familiarity exceeds 0.7 within the first few thousand queries, so a single agent often meets τ by itself, and the router mostly receives solo updates, which pull a head closer whenever its reward is positive, regardless of how it compares with other agents. The margin condition also fails: $Ne^{-\lambda m} = 4e^{-1} \approx 1.47 > \tau$, so even agents pushed to the margin jointly clear the threshold, and the full-pool fallback is never triggered. Familiarity then stops distinguishing agents, and training data is no longer steered toward emerging specialists.

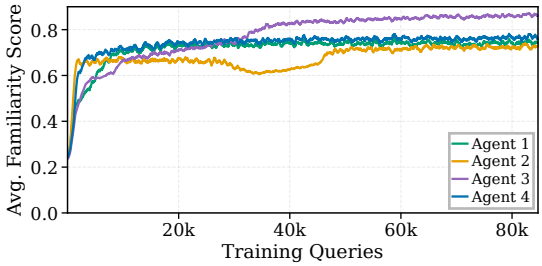
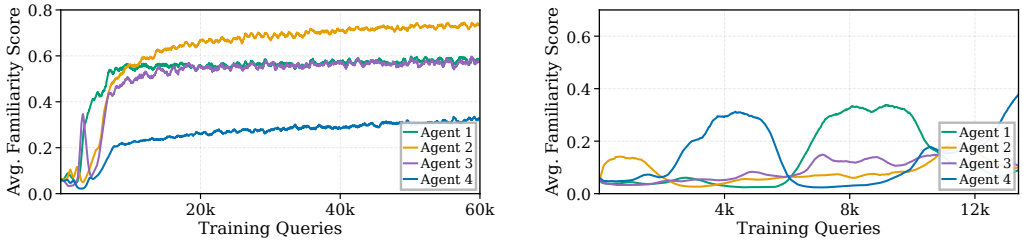


Figure 6: Average familiarity score per agent during training with $\lambda = 1.0$.

Separation margin m. We compare the default margin $m = 1.0$ with a larger margin $m = 2.0$ and with no margin, i.e., using $A_{i,q}d_i(q)$ for negative advantages as well. With $m = 2.0$, accuracy drops to 59.2 ID and 70.7 OOD at a similar cost (364.74 tokens). Figure 7a shows that Agent 4's familiarity stays near 0.3 while the other agents reach 0.55–0.75, so one agent is rarely selected: a larger margin lets negative advantages push a head down to $e^{-\lambda m} \approx 0.02$, from which it recovers slowly. Without a margin, the repulsive term never switches off, and pushes from negative advantages keep cancelling pulls from positive ones. Figure 7b shows that all familiarity scores then oscillate between about 0.03 and 0.35 and none stays high in the run shown. Both results support the role of the margin described in Appendix A.1.



(a) Margin $m = 2.0$.

(b) No margin.

Figure 7: Average familiarity score per agent during training with a larger margin and without a margin.

E.4 Ablation on Routing and Exploration

We ablate two training-time design choices on the default pool (Table 10): replacing cumulative-threshold allocation with fixed Top-3 allocation during training, and removing both exploration bonuses ($c_{\text{ucb}} = w_{\text{ent}} = 0$). All variants use cumulative-threshold routing at test time.

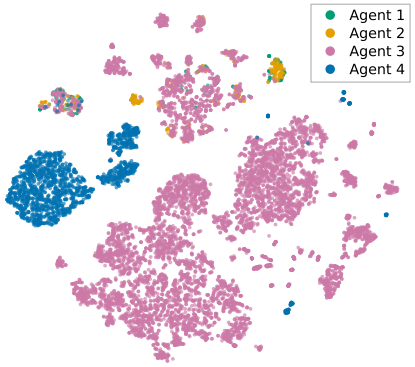
Table 10: Ablation of training-time allocation and exploration.

Training Configuration	**ID Avg.**	**OOD Avg.**	**Avg. Tokens**
Fixed Top-3 allocation	63.1	72.3	817.97
No exploration (w/o UCB & entropy)	57.1	63.9	436.60
Cumulative threshold (ours)	63.2	72.8	367.77

Fixed Top-3 allocation. Training with a fixed Top-3 allocation gives similar accuracy but more than doubles the test-time token count (817.97 vs. 367.77). Figure 8a shows that familiarity never becomes a reliable signal under this scheme. Two agents stay low throughout, one near its initial value of about 0.06 and one near $e^{-\lambda m} \approx 0.14$, the level at which the margin stops repulsion, while the other two alternate between about 0.15 and 0.55. Because three agents are activated on every training query regardless of fit, familiarity is never tied to a small set of reliable specialists. At test time, the summed familiarity therefore often stays below τ, and many queries fall back to all four agents.



(a) Familiarity scores under fixed Top-3 training.



(b) t-SNE of routed queries without exploration.

Figure 8: Training-time ablations. (a) Average familiarity score per agent when every training query activates three agents. (b) Mid-layer features of training queries, colored by the agent they are routed to, when both exploration bonuses are removed.

Exploration bonuses. Removing the UCB and entropy bonuses lowers accuracy by 6.1 ID and 8.9 OOD points. The t-SNE plot in Figure 8b shows that most queries are routed to Agent 3, Agent 4 keeps a few clusters, and Agents 1 and 2 receive almost no queries. Without the bonuses, whichever agent gains familiarity first receives more data, improves further, and becomes even more familiar, leaving the other agents without training data. The UCB term guarantees that under-used agents are tried, and the entropy term favors agents whose completions are still diverse.

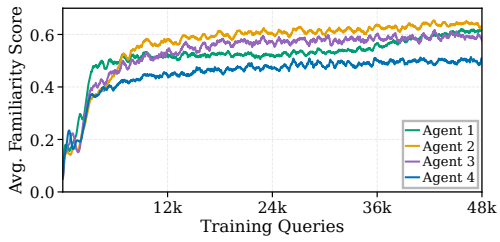
E.5 Ablation on Hidden-State Depth

We replace the mid-layer features of Eq. 2 with the final-layer hidden state of the last non-padding token and keep everything else fixed. Table 11 shows that final-layer features lower accuracy by 2.1 ID and 1.5 OOD points and increase generated tokens by 19%.

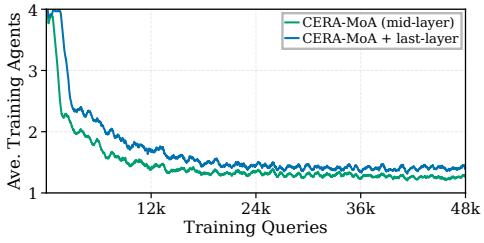
Table 11: Ablation of the hidden-state layer used by the router.

Feature Source	ID Avg.	OOD Avg.	Avg. Tokens
Final-layer states	61.1	71.3	438.12
Mid-layer states (ours)	63.2	72.8	367.77

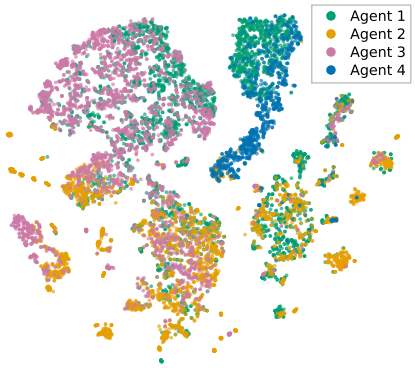
Figure 9 explains the gap. Final-layer familiarity rises more slowly early in training than mid-layer familiarity (Figure 9a vs. Figure 5c), so the router has to activate more agents per training query to reach the same threshold, and it keeps doing so throughout training (Figure 9b). Each agent therefore receives a less targeted mix of queries. In the t-SNE plot of final-layer features (Figure 9c), queries routed to different agents overlap heavily, whereas mid-layer features give well-separated clusters (Figure 3, right). This agrees with Skean et al. (2025), who find that intermediate layers often give better representations than the final layer, which is specialized for next-token prediction.



(a) Familiarity scores with final-layer features.



(b) Activated agents per training query.



(c) t-SNE of final-layer features.

Figure 9: Routing with final-layer features. (a) Average familiarity score per agent. (b) Average number of agents activated per training query with mid-layer and final-layer features. (c) Final-layer features of training queries, colored by the agent they are routed to.

F LIMITATIONS

Prompt-level routing. Familiarity is computed once from the initial prompt, so CERA-MoA is designed for single-turn tasks. In multi-turn workflows, such as iterative software development or multi-agent debate, the required expertise may change as the context grows. The estimator itself applies to any context: recomputing $h(\cdot)$ on the current context at each turn is a direct extension, but training the router from per-turn rewards remains to be studied.

Single-agent output on open-ended tasks. For code, CERA-MoA returns the completion of the most familiar activated agent instead of combining candidates. This keeps inference cheap but discards the other activated agents' completions, which may explain part of the gap to the strongest baselines on some code benchmarks (e.g., MBPP and HumanEval). A generative aggregator, such as an LLM meta-thinker that synthesizes the selected agents' outputs, can replace this rule without changing routing or training.

Verifiable rewards. Both the agents and the router are trained from rule-based rewards. Tasks without automatic verifiers would require a learned reward model or judge, whose errors would propagate into routing as well as into the policies.