PARTIROUTE: DECENTRALIZED SELF-PARTICIPATION FOR MULTI-LLM ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) routing is commonly formulated as a centralized selection problem, often through a routing decision that selects a single model or determines a routing path. Such early commitment can discard useful alternative responses and limit recovery from suboptimal routing decisions. We argue that multi-LLM routing should instead separate candidate construction from final-answer selection, reformulating the question from which model should be selected to whether each model should participate. Based on this view, we introduce PartiRoute, a decentralized routing framework in which each frozen LLM is paired with an independently parameterized policy that makes a model-wise participation decision. Their joint decisions form a query-dependent candidate pool optimized under a shared quality–cost utility. To capture the set-dependent contribution of each model, PartiRoute uses an action-flip difference reward for model-specific credit assignment and trains the routing policies directly with PPO, without fine-tuning the candidate LLMs or requiring supervised router warm-up. Experiments across five benchmarks and 13 heterogeneous LLMs show that PartiRoute forms dynamic candidate pools while achieving competitive accuracy–cost performance. Response-aware aggregation further improves the utilization of retained responses, highlighting the value of decentralized self-participation and the separation between candidate construction and final-answer aggregation.

1 INTRODUCTION

Large language models (LLMs) differ substantially in capability, specialization, and inference cost, and no single model is uniformly optimal across queries (Hu et al., 2024; Varangot-Reille et al., 2025). Always using the strongest model can incur unnecessary computation, while relying on cheaper models may sacrifice answer quality on more difficult queries. LLM routing therefore aims to allocate model capacity according to the incoming query, improving the trade-off between response quality and inference cost. Related work on test-time scaling similarly suggests that the value of additional inference computation depends on query difficulty (Snell et al., 2025). Existing systems commonly use single-model selection or LLM cascades, which progressively escalate queries to stronger models when needed (Ong et al., 2025; Chen et al., 2023; Aggarwal et al., 2024). This raises a natural question: *Should routing be required to determine the responder before alternative model responses are available?*

A key limitation of these routing interfaces is that candidate construction and final-answer selection are coupled within the same routing decision.In single-model routing, the selection of one model simultaneously determines which response is generated and returned (Ong et al., 2025; Feng et al., 2025). LLM cascades allow sequential escalation, but skipped models still contribute no response to downstream decision making (Chen et al., 2023; Aggarwal et al., 2024; Zellinger et al., 2025). Once a model is excluded, its response is unavailable for downstream comparison or aggregation, making routing errors difficult to recover from. This coupling conflates two decisions: which responses should be made available and which available response should ultimately be returned. Routing therefore need not immediately determine the final response model; it can first determine which model responses should be made available and only then decide which response should ultimately be returned. Figure 1 contrasts single-model routing, DiSRouter-style path routing (Zheng et al., 2026), and PartiRoute's model-wise candidate construction.

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.

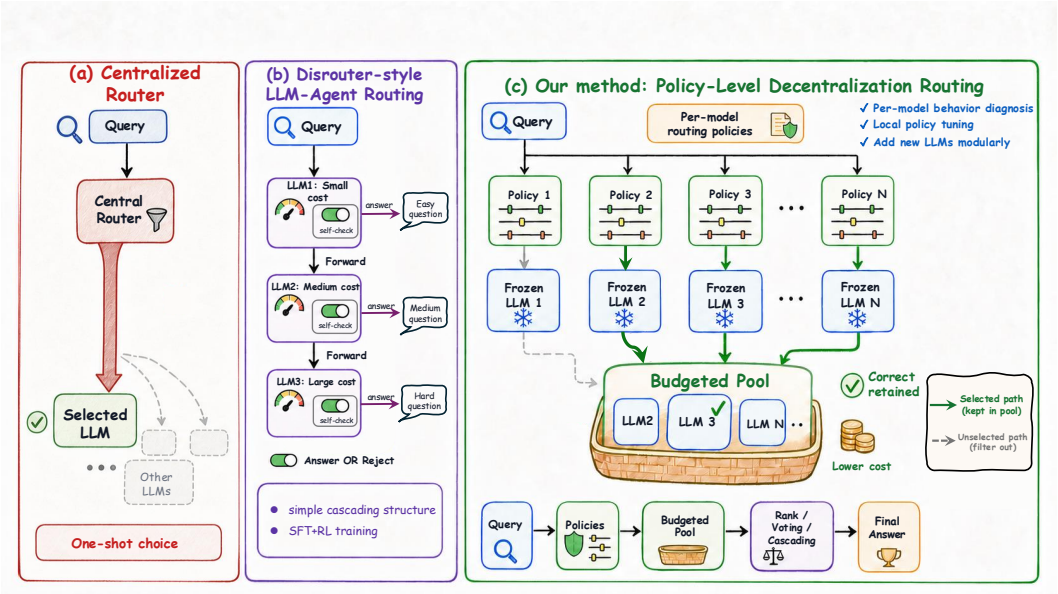


Figure 1: Three routing interfaces. (a) A centralized router commits to one model; (b) a self-routing cascade traverses a model path; (c) our model-wise policies construct a budgeted candidate pool whose responses are resolved by a downstream decision rule. The figure is conceptual: claims about adding unseen LLMs require separate evaluation and are not made in this paper.

Motivated by this limitation and the separation, we introduce **PartiRoute**, a decentralized self-participation framework for multi-LLM routing. We reformulate multi-LLM routing as *model-wise participation*: given a query and a pool of candidate LLMs, each model is paired with an independently parameterized policy that makes a binary decision about whether it should participate. The joint participation decisions form a query-dependent candidate pool, whose size and composition may vary across queries. Only the selected models generate responses, which are then processed by a downstream aggregation rule to determine the final answer. Rather than selecting a single winner at the routing stage, PartiRoute separates the construction of available responses from the subsequent selection of the response to return.

This reformulation introduces a nontrivial credit-assignment problem because the value of one model's participation depends on which other models are selected for the same query. A model may be useful when selected alone but redundant when a comparable response is already available, while another model may provide complementary value only for particular queries. We address this set-dependent interaction with model-specific Actor–Critics and an action-flip difference reward that measures each model's marginal contribution while holding the remaining participation decisions fixed. We additionally study a lightweight same-tier residual gate to examine whether restricted peer information can further improve coordination among the local policies.

Experiments across five benchmarks and 13 heterogeneous LLMs show that PartiRoute constructs query-dependent candidate sets while maintaining competitive accuracy–cost performance against representative routing baselines. Strong candidate coverage preserves useful answer opportunities, but final-answer quality still depends on how the retained responses are resolved downstream. As an additional extension, response-aware aggregation provides stronger signals for final-answer selection than routing probabilities alone, enabling the system to make more effective use of the retained responses. Overall, PartiRoute maintains strong final-answer performance while controlling execution cost, supporting the separation between candidate construction and final-answer selection as a practical way to preserve useful response opportunities without sacrificing the performance–cost balance.

This work makes three main contributions:

- **Decentralized model-wise self-participation.** To the best of our knowledge, PartiRoute is the first LLM routing framework built around decentralized model-wise self-participation. Each model has an independent local policy that decides whether it should participate, and the joint decisions form a query-dependent candidate set.
- **Credit assignment for model-wise participation.** We introduce an action-flip difference reward for model-specific credit assignment. It measures each model's marginal contribution while holding the other participation decisions fixed, providing a direct learning signal for the local policies.
- **Reinforcement-learning-only policy optimization.** We optimize PartiRoute's trainable routing policies directly with PPO (Schulman et al., 2017), without SFT/BCE warm-up, or fine-tuning the candidate LLMs.

2 Related Work

Cost-aware routing and response aggregation. LLM routing balances response quality against inference cost. RouteLLM learns strong-versus-weak model selection from preference data (Ong et al., 2025), while GraphRouter predicts query–model quality and cost through a heterogeneous graph (Feng et al., 2025). FrugalGPT and AutoMix instead use cascades to determine whether additional model execution is warranted (Chen et al., 2023; Aggarwal et al., 2024). Beyond single-model selection, C2MAB-V studies cost-aware combinatorial multi-LLM selection with online feedback (Dai et al., 2024). LLM-Blender ranks and fuses multiple model outputs (Jiang et al., 2023), and self-consistency aggregates sampled reasoning paths (Wang et al., 2023). These studies establish both multi-model selection and response aggregation as existing directions. PartiRoute focuses on query-dependent candidate construction through model-specific participation policies, with candidate coverage evaluated separately from downstream final-answer quality.

Distributed and model-wise routing. DiSRouter distributes answer-or-forward decisions across LLM agents, allowing a query to traverse a model network according to learned self-awareness (Zheng et al., 2026). AgentNet studies decentralized task routing and evolving connectivity among LLM agents (Yang et al., 2025). PartiRoute differs from DiSRouter in both the decision mechanism and routing output: external policies associated with frozen LLMs make binary participation decisions, whose joint outcome is a candidate set rather than a traversal path. Decentralization in PartiRoute therefore refers to the model-specific decision policies rather than fully decentralized training. The separate policies are trained using a shared set utility, and their selected responses are processed by a downstream decision rule.

Policy learning and credit assignment. PartiRoute uses PPO with independently parameterized Actors and Critics, following the independent-learning structure studied by de Witt et al. (2020) and the clipped policy objective of Schulman et al. (2017). Counterfactual credit assignment evaluates an agent's action while holding other agents' actions fixed. COMA implements this principle using a centralized critic and an action-marginalized baseline (Foerster et al., 2018). PartiRoute instead compares the candidate utility before and after flipping one binary participation action. Both utilities are computed directly from the offline response records, so the action-flip difference for the stored responses does not require a learned centralized critic.

3 Problem Formulation

We formalize centralized single-model routing and model-wise participation under a common quality–cost objective.

3.1 General Routing Problem

What should a routing system ultimately optimize? Let $x \sim \mathcal{D}$ denote a query and let $\mathcal{M} = \{m_1, \ldots, m_N\}$ be a pool of candidate LLMs. Regardless of how routing decisions are organized, the system must balance final-answer quality against inference cost. We therefore evaluate routing methods through the common objective

$$\text{Utility} = \text{Performance} - \alpha\ \text{Cost}, \tag{1}$$

where $\alpha \geq 0$ controls the relative importance of answer quality and inference cost.

3.2 Centralized Single-Model Routing

A centralized router uses a global policy π_c to map each query to one model in the candidate pool: $\pi_c(x) \in \mathcal{M}$. For model m_i, let $q_i(x)$ denote its response quality for query x, and let $c_i(x)$ denote its inference cost. Under the common quality–cost objective, the optimal centralized policy is

$$\pi_c^* = \arg\max_{\pi_c} \mathbb{E}_{x\sim\mathcal{D}} \left[q_{\pi_c(x)}(x) - \alpha c_{\pi_c(x)}(x) \right]. \tag{2}$$

For each query, only the selected model $\pi_c(x)$ is executed and its response is returned as the final answer.

3.3 Model-Wise Self-Participation Routing

PartiRoute represents candidate construction through a collection of model-wise participation policies $\Pi = \{\pi_1, \ldots, \pi_N\}$. Each independently parameterized policy $\pi_i(\cdot; \theta_i)$ produces a binary participation action $a_i(x) \in \{0, 1\}$ indicating whether model m_i participates. Their joint decisions form the query-dependent candidate pool

$$S(x) = \{m_i \in \mathcal{M} : a_i(x) = 1\}. \tag{3}$$

Both the size and composition of $S(x)$ may vary across queries. Candidate construction balances retained response quality against the cost of executing the selected models through

$$U_{\text{cand}}(S, x) = \max_{m_i \in S} q_i(x) - \lambda \sum_{m_i \in S} c_i(x), \tag{4}$$

where $\lambda \geq 0$ controls the cost penalty. This candidate-level objective is distinct from the system-level Utility in Eq. equation 1.

The participation policies are optimized by maximizing the expected candidate utility:

$$\Pi_{\text{cand}}^* = \arg\max_{\Pi} \mathbb{E}_{x\sim\mathcal{D}} \left[U_{\text{cand}}(S(x), x) \right]. \tag{5}$$

During training, the expectation also includes participation actions sampled from the local policies.

Once the candidate pool is constructed, only the selected models generate responses. Let $r_i(x)$ denote the response produced by model m_i. A downstream decision rule then produces

$$\hat{a}(x) = g\left(x, \{r_i(x) : m_i \in S(x)\}\right), \tag{6}$$

where g denotes the downstream final-answer selection rule.

Because U_{cand} evaluates the candidate pool jointly, Eq. equation 5 does not by itself identify the contribution of each individual participation decision. This set-dependent credit-assignment problem motivates the learning mechanism introduced in the next section.

4 PartiRoute

4.1 Overview

This section presents PartiRoute, a model-wise participation framework for dynamic candidate routing. The model-wise formulation in Problem Formulation replaces a single routing decision with

a set of binary participation decisions. This raises a key learning problem: *How should each local policy be trained when its value depends on the other selected models?*

PartiRoute pairs each frozen LLM with an independently parameterized Actor–Critic participation policy. An action-flip difference reward assigns model-specific credit by reversing one participation decision while holding the others fixed and measuring the resulting utility difference. The joint decisions form the candidate pool in Eq. equation 3; only selected models generate responses for downstream final-answer selection. An optional same-tier residual gate refines participation probabilities using restricted peer summaries. Figure 2 summarizes the framework.

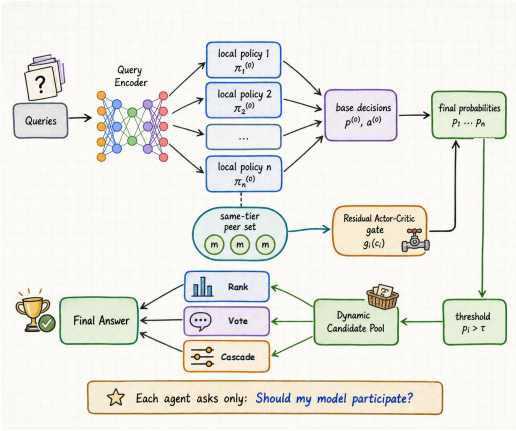


Figure 2: **PartiRoute inference pipeline.** Model-specific policies produce local participation probabilities, with optional same-tier residual coordination before dynamic candidate-set construction. Selected models generate responses for downstream aggregation.

4.2 Model-Specific Participation Policies

How does each model form its own participation decision?

PartiRoute instantiates each participation policy π_i with a model-specific Actor whose parameters are not shared with other models. For model m_i, the Actor receives a local observation combining the query representation with model-specific information:

$$o_i^{(0)} = [e_x; e_i; z_i^{\text{local}}; z_i^{\text{relative}}], \tag{7}$$

where e_x represents the query, e_i is a learned model embedding, z_i^{local} describes the model's own characteristics, and z_i^{relative} captures static differences from reference and same-tier models.

The Actor maps this observation to an initial participation probability $p_i^{(0)} = \sigma(\ell_i^{(0)})$, which determines the model's participation action $a_i(x)$. The joint actions instantiate the candidate pool $S(x)$ in Eq. equation 3.

The base local policy does not observe other models' current participation decisions or responses, so its initial decision is formed locally from the query and model-specific information. These local actions jointly instantiate the candidate pool $S(x)$ in Eq. equation 3, making the value of each participation decision dependent on the other selected models.

4.3 Set-Dependent Credit Assignment

How should each local policy be trained when its value depends on the other selected models?

The candidate utility in Eq. equation 4 evaluates the selected models jointly. A shared utility alone, however, does not reveal the contribution of each participation decision. A model may improve the candidate pool when it contributes a better response, but become redundant when a comparable response is already selected.

We therefore assign each local policy a counterfactual learning signal by flipping only its participation action while holding all others fixed. Let $\mathbf{a}$ denote the joint participation actions, and let $\mathbf{a}^{\mathrm{flip}(i)}$ differ from $\mathbf{a}$ only in a_i. The difference reward for model i is

$$D_i(x, \mathbf{a}) = U_{\mathrm{cand}}(S(x; \mathbf{a}), x) - U_{\mathrm{cand}}\Big(S(x; \mathbf{a}^{\mathrm{flip}(i)}), x\Big). \tag{8}$$

A positive D_i indicates that the model's decision improves the candidate utility, while a negative value indicates that it reduces the utility. Because the offline response records provide the quality and cost of each model, both utilities can be computed directly. This gives an exact action-flip signal without estimating the counterfactual value with a centralized critic (Foerster et al., 2018).

4.4 PPO Optimization of Local Policies

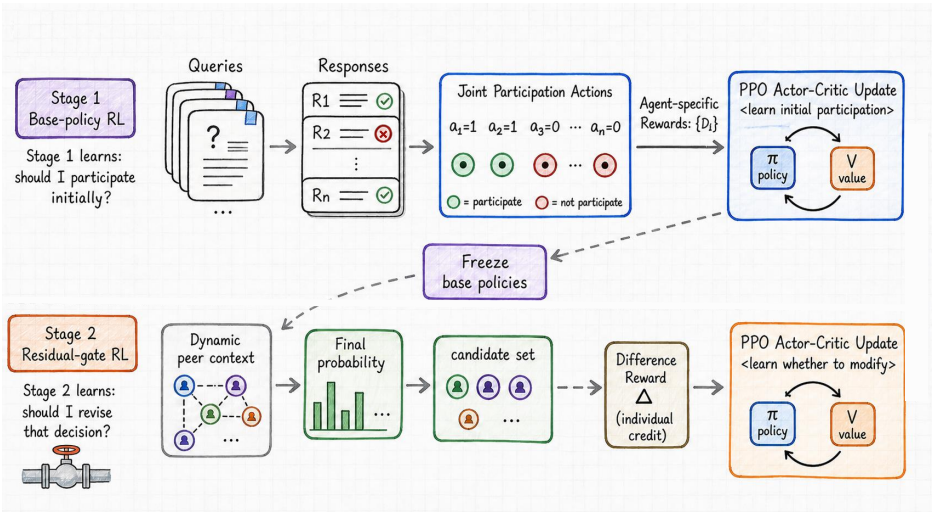


Figure 3: Sequential training. Stage one learns base participation policies; stage two freezes them and learns residual gates from dynamic peer context. Both stages use model-specific PPO Actor–Critics and set-difference rewards.

How are the local policies optimized from their model-specific rewards?

The difference reward in Eq. equation 8 provides a model-specific learning signal for each participation policy. We pair each Actor with a model-specific Critic and optimize the resulting Actor–Critic with PPO (Schulman et al., 2017; de Witt et al., 2020). The Actor parameterizes the binary participation decision through $p_i^{(0)}$, while the Critic estimates the expected difference reward for policy optimization.

Each Actor–Critic is optimized independently while all candidate LLMs remain frozen. Quality and cost are used only to compute the candidate utility and action-flip reward after participation actions are sampled, rather than as Actor inputs. We train the local policies directly with PPO without supervised router warm-up or LLM fine-tuning. We additionally study an optional same-tier residual coordination stage, termed PartiRoute-Peer, which refines the frozen base participation probabilities using restricted peer-policy information. Figure 3 summarizes the two-stage training procedure, while PPO details and the Peer formulation are provided in Appendix A.1 and Appendix A.2, respectively.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We evaluate PartiRoute on five benchmarks: ARC-Challenge (Clark et al., 2018), GSM8K (Cobbe et al., 2021), MBPP (Austin et al., 2021), MMLU (Hendrycks et al., 2021), and OpenBookQA (Mihaylov et al., 2018). The candidate pool contains 13 instruction-tuned models from the Gemma, Mistral, and Qwen2.5 families. All consolidated router comparisons and threshold sweeps use a validation split of 2,920 queries; we therefore treat the reported results as validation evidence.

We report final-answer accuracy, normalized response-token cost, and $\mathrm{Utility} = \mathrm{Accuracy} - \alpha\,\mathrm{Cost}$ with $\alpha = 0.5$. Candidate-level diagnostics include candidate hit rate, defined as the fraction of queries whose candidate pool contains at least one correct response, and mean candidate-set size. We compare against single-model and random baselines, a non-deployable oracle, RouteLLM-MF (Ong et al., 2025), FrugalGPT-style routing (Chen et al., 2023), GraphRouter (Feng et al., 2025), and DiSRouter (Zheng et al., 2026). Full model specifications, cost normalization, baseline protocols, training settings, seeds, and hardware details are provided in Appendix B and Appendix A.

Implementation details. The base policies are trained with PPO for 200 epochs using batch size 64 and a learning rate of 3×10^{-4}. The residual gates are then trained for 300 epochs with batch size 128 and a learning rate of 10^{-3}, while the base policies remain frozen. Experiments are conducted on eight NVIDIA RTX A6000 GPUs (48 GB each).

5.2 OVERALL ACCURACY–COST PERFORMANCE

Baseline comparison. We compare PartiRoute with single-model selection rules and representative routing baselines, including RouteLLM-MF (Ong et al., 2025), a FrugalGPT-style router (Chen et al., 2023), GraphRouter (Feng et al., 2025), and DiSRouter (Zheng et al., 2026). As shown in Table 1, PartiRoute-Local achieves competitive accuracy–cost performance relative to these baselines, while PartiRoute-Peer performs similarly without an additional gain at this operating point. Overall, the results show that PartiRoute remains competitive with representative routing methods while retaining multiple candidate responses for downstream aggregation.

Table 1: **Comparison with multiple routing methods.** PartiRoute-Local is averaged over three seeds, whereas PartiRoute-Peer uses one seed. Oracle is shown as a non-deployable reference.

Group	Method	Accuracy ↑	Cost ↓	Utility ↑
Topline	Oracle	0.977	0.125	0.915
Naive	Largest model	0.820	0.287	**0.677**
	Smallest model	0.359	0.578	0.070
	Random	0.569	0.427	0.356
Routers	RouteLLM-MF	0.818	0.282	**0.677**
	FrugalGPT-style	0.687	0.250	0.562
	GraphRouter	0.811	0.294	0.664
	DiSRouter	0.651	0.175	0.563
Ours	PartiRoute-Local	**0.821**	0.290	0.676
	PartiRoute-Peer	0.819	0.289	0.675

Accuracy–cost trade-off analysis. To move beyond the single operating point in Table 1, we vary the participation threshold to examine how routing behavior changes across different cost and candidate-pool budgets.We compare PartiRoute with multiple routing baselines across different operating points.Figure 4 shows that expanding the candidate pool generally improves candidate hit rate, but does not lead to a corresponding improvement in final-answer accuracy. This gap indicates that retaining a correct response is not sufficient to guarantee a correct final prediction; the benefit of larger candidate pools depends on how the retained responses are subsequently aggregated. The participation threshold therefore provides a controllable accuracy–cost operating point, while

downstream answer selection determines how effectively the retained candidates are converted into final-answer quality.

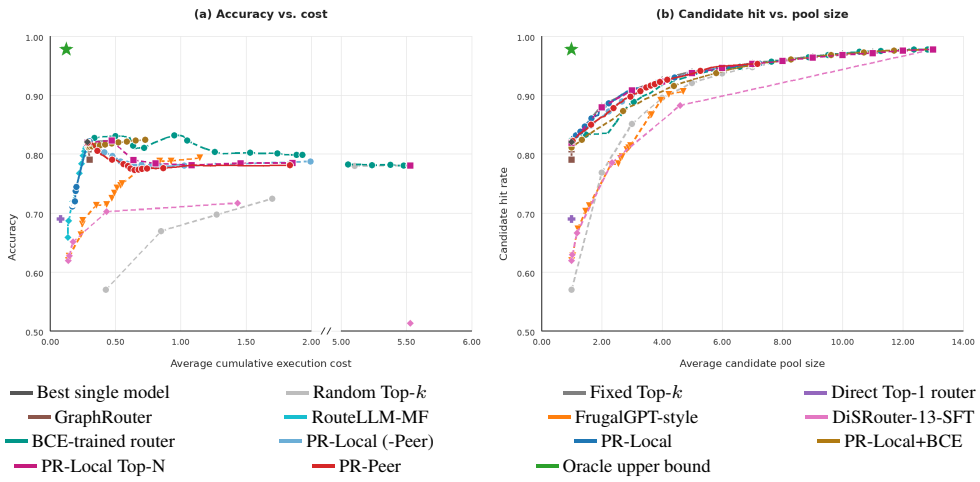


Figure 4: **Threshold-dependent accuracy–cost and candidate-coverage trade-offs.** Panel (a): Accuracy vs. Cost. Panel (b): Candidate Hit vs. Pool Size. PR denotes PartiRoute in the legend.

5.3 From Candidate Coverage to Final-Answer Selection

Final-response ROC-AUC analysis. Retaining a correct response in the candidate set does not guarantee that it will be chosen as the final answer. Motivated by the gap between candidate coverage and final-answer accuracy in Figure 4, we examine whether participation probabilities alone provide a reliable signal for answer selection, or whether the candidate responses themselves are needed. Figure 5 compares policy-based signals with response-aware aggregation strategies using ROC-AUC. Response-aware voting consistently provides stronger discrimination between correct and incorrect final decisions than signals derived only from routing probabilities, with vote-2 showing the strongest overall result among the tested aggregation strategies. These findings reinforce the separation between candidate construction and final-answer selection: the routing policies determine which responses are retained, while response-level agreement is more informative for deciding which answer to return.

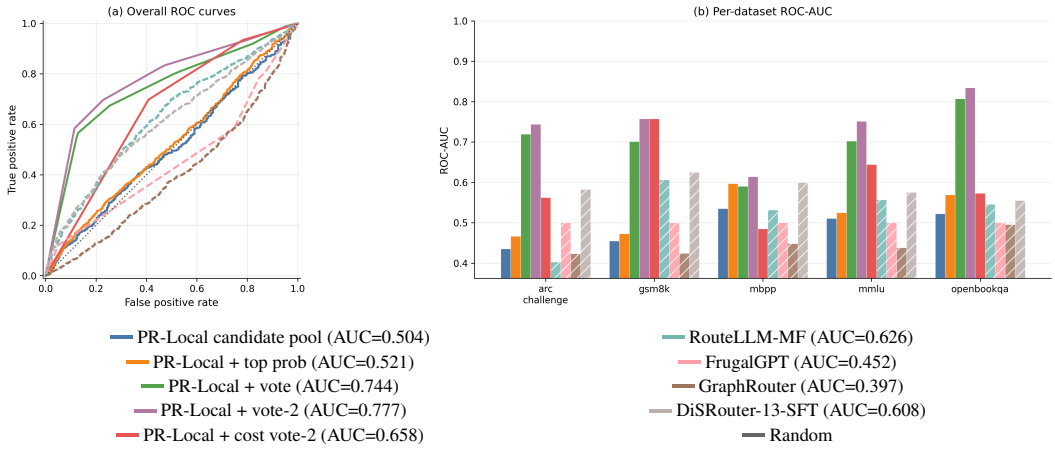


Figure 5: **ROC analysis of candidate and answer-aggregation signals.** Overall curves are shown on the left and per-dataset AUC values on the right. PR abbreviates PartiRoute in the legend.

5.4 Dynamic Candidate-Set Analysis

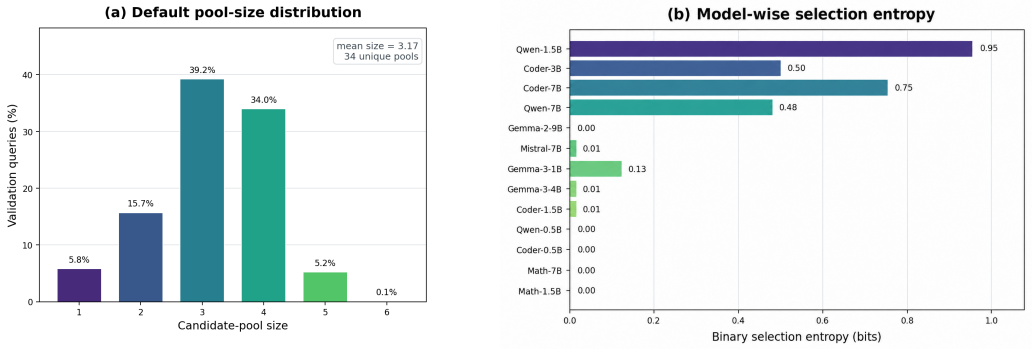


Figure 6: **Dynamic candidate-set analysis of PartiRoute-Local.**

Candidate-set size. Candidate-set size. PartiRoute does not rely on a fixed Top-k candidate pool, but instead varies the number of selected models across queries, as reflected in Figure 6(a). Model inclusion rates also differ across datasets, with some models selected consistently and others showing more dataset-dependent participation. These results indicate that PartiRoute adapts both the size and composition of the candidate pool.

Model-wise participation variability. Dataset-wise inclusion rates summarize average participation, but they do not capture how consistently each model is selected across individual queries. We measure participation variability using binary selection entropy, $H_m = -q_m \log_2 q_m - (1 - q_m) \log_2(1 - q_m)$, where q_m is the inclusion rate of model m. Higher entropy indicates greater variation in participation. Figure 6(b) shows that model participation varies in stability across queries. Selection entropy captures this variation rather than decision correctness.

6 Conclusion and Discussion

Conclusion. We presented PartiRoute, a dynamic candidate routing framework based on decentralized model-wise self-participation. Instead of committing each query to a single model at the routing stage, independently parameterized local policies decide whether each model should participate. Their joint decisions form a query-dependent candidate pool under a shared quality–cost objective, while action-flip rewards provide model-specific credit assignment. Experiments across five benchmarks and 13 heterogeneous LLMs show that PartiRoute can construct dynamic candidate pools while maintaining competitive accuracy–cost performance. The results also highlight the distinction between retaining useful responses and selecting the final answer, making downstream aggregation an important part of the overall routing pipeline.

Limitations. One remaining challenge is that dynamic candidate construction may execute multiple models for a single query, leaving further room to reduce inference cost without sacrificing candidate quality. In addition, strong candidate coverage does not necessarily translate into equally strong final-answer quality, suggesting that the retained responses could be exploited more effectively by downstream aggregation. Finally, PartiRoute is decentralized at the level of model-specific participation policies, but the current training procedure still relies on a shared candidate utility and centrally available offline response records. Fully decentralized training and credit assignment therefore remain open directions.

Future Work. Future work can explore more budget-aware participation policies, adaptive candidate-pool sizing and early stopping, and richer aggregation or verification mechanisms that make better use of agreement, disagreement, and complementary information among candidate responses. Another direction is to jointly optimize candidate construction and final-answer selection, while developing more decentralized training procedures that reduce reliance on globally available response and utility information. It is also promising to extend PartiRoute to larger and dynamically changing model pools.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript organization, English-language editing and polishing, and LaTeX formatting. Generative AI was not used to generate experimental measurements or to modify reported empirical results. All AI-assisted edits and suggestions were reviewed by the authors against the underlying source material and manuscript. The authors take full responsibility for the final content of the paper.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. Automix: Automatically mixing language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 131000–131034, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ecda225cb187b40ea8edc1f46b03ffda-Abstract-Conference.html.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance, 2023. URL https://arxiv.org/abs/2305.05176.

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. URL https://arxiv.org/abs/1803.05457.

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

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

Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip H. S. Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge?, 2020. URL https://arxiv.org/abs/2011.09533.

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

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi: 10.1609/aaai.v32i1.11794. URL https://ojs.aaai.org/index.php/AAAI/article/view/11794.

Gemma Team. Gemma 2: Improving open language models at a practical size, 2024. URL https://arxiv.org/abs/2408.00118.

Gemma Team. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. RouterBench: A benchmark for multi-LLM routing system, 2024. URL https://arxiv.org/abs/2403.12031.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 14165–14178. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.792. URL https://aclanthology.org/2023.acl-long.792/.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

Mistral AI. Mistral-7b-instruct-v0.2. Hugging Face model card, 2023. URL https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2.

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

Qwen Team. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b623663fd9b874366f3ce019fdfdd44-Abstract-Conference.html.

Clovis Varangot-Reille, Christophe Bouvard, Antoine Gourru, Mathieu Ciancone, Marion Schaeffer, and François Jacquenet. Doing more with less: Implementing routing strategies in large language model-based systems: An extended survey, 2025. URL https://arxiv.org/abs/2502.00409.

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

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. AgentNet: Decentralized evolutionary coordination for LLM-based multi-agent systems. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3578. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/9a379c1b05793d1c42dc832269834515-Abstract-Conference.html.

Michael J. Zellinger, Rex Liu, and Matt Thomson. Cost-saving LLM cascades with early abstention, 2025. URL https://arxiv.org/abs/2502.09054.

Hang Zheng, Hongshen Xu, Yongkai Lin, Shuai Fan, Lu Chen, and Kai Yu. DiSRouter: Distributed self-routing for LLM selections. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=KDcwXKr0NU.

A Optimization Details

A.1 PPO Objective

For agent i, the Critic estimates $V_i(c_i)$ and the advantage is $A_i = D_i - V_i(c_i)$. With ratio

$$\rho_i = \frac{\pi_i(a_i \mid c_i)}{\pi_i^{\text{old}}(a_i \mid c_i)}, \tag{9}$$

the clipped Actor loss, Critic loss, and combined loss are

$$\mathcal{L}_{\text{actor}}^{(i)} = -\mathbb{E}\left[\min\left(\rho_i A_i, \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon)A_i\right)\right], \tag{10}$$

$$\mathcal{L}_{\text{critic}}^{(i)} = \mathbb{E}[(V_i(c_i) - D_i)^2], \tag{11}$$

$$\mathcal{L}_i = \mathcal{L}_{\text{actor}}^{(i)} + \alpha_v \mathcal{L}_{\text{critic}}^{(i)} - \eta H(\pi_i). \tag{12}$$

A.2 Optional Same-Tier Residual Coordination

We additionally examine whether restricted information from other local policies can improve coordination without introducing a centralized router. This extension, referred to as PartiRoute-Peer, refines the participation probabilities produced by the frozen base local policies using model-specific residual gates.

Same-tier peer context. For model m_i, let $\mathcal{P}_i$ denote the set of models in the same tier. After the base local policies produce their initial participation probabilities and temporary decisions, the peer context is constructed as

$$c_i = \text{summary}\left(p_i^{(0)}, a_i^{(0)}, \{p_j^{(0)}, a_j^{(0)}\}_{j \in \mathcal{P}_i}\right). \tag{13}$$

The summary is fixed-dimensional and contains only restricted within-tier information, including participation-probability statistics, temporary inclusion information, candidate-set size, and cost summaries. It does not contain peer responses, correctness labels, query-level outcome scores, or cross-tier dynamic information.

This restriction preserves the model-specific structure of the routing policies: peer information only refines an already formed local participation decision rather than constructing a centralized routing decision.

Residual probability adjustment. For model m_i, a model-specific residual gate predicts

$$\Delta\ell_i = f_{\phi_i}(c_i)$$

and adjusts the frozen base probability $p_i^{(0)}$ as

$$p_i = \sigma\left(\text{logit}\left(\text{clip}(p_i^{(0)}, 0.05, 0.95)\right) + \Delta\ell_i\right). \tag{14}$$

The clipping avoids extreme logits, while the residual head is zero-initialized so that the second stage begins near the frozen base policy.

Residual-gate optimization. The residual stage is trained only after the base local policies have converged. The base-policy parameters are frozen, and the residual-adjusted probabilities determine the participation actions used to construct the candidate pool.

Training uses the same candidate utility in Eq. equation 4 and the same action-flip difference reward in Eq. equation 8. Thus, each residual gate is optimized from the corresponding model-specific signal D_i, while all base-policy parameters remain fixed.

The resulting optimization procedure is sequential rather than end-to-end:

$$\underbrace{\{\theta_i\}_{i=1}^{N}}_{\text{base local policies}} \longrightarrow \text{freeze} \longrightarrow \underbrace{\{\phi_i\}_{i=1}^{N}}_{\text{residual gates}}. \tag{15}$$

A.3 Optimization Settings

The base local policies are trained for 200 outer epochs using batches of 64 query groups. Each batch uses four PPO update epochs with learning rate 3×10^{-4}. The PPO clipping coefficient is $\epsilon = 0.2$, the critic coefficient is $\alpha_v = 0.5$, and the entropy coefficient is $\eta = 0.01$.

For the same-tier residual stage, the frozen base policies are retained while the residual gates are trained for 300 outer epochs using batches of 128 query groups, four PPO update epochs per batch, and learning rate 10^{-3}.

B Experimental Details

B.1 Evaluation Protocol and Metrics

For query x from dataset d, response-token cost is normalized as

$$c_i(x) = \min\left(\frac{\text{output_tokens}_i(x)}{\text{P95}_{\text{train},d}}, 1\right). \tag{16}$$

Candidate cost is the sum of $c_i(x)$ over selected models and may therefore exceed one. System utility is computed as $\text{Accuracy} - 0.5\,\text{Cost}$. Candidate hit rate measures whether the selected candidate pool contains at least one correct response.

B.2 Model Pool

Table 2 lists the 13 locally deployed instruction-tuned checkpoints from the Gemma, Mistral, and Qwen2.5 families (Gemma Team, 2024; 2025; Mistral AI, 2023; Qwen Team, 2024). Parameter counts follow the released model variants. Context denotes the native maximum sequence length configured by each released checkpoint; it is distinct from the generated response-token count used as routing cost in Equation 16. No commercial API was used to generate the response records.

Table 2: Model pool. Native context is the checkpoint-configured maximum sequence length without optional context-extension methods.

Checkpoint	Parameters	Native context (tokens)	Primary focus
Gemma-2-9B-IT	9B	8,192	General
Gemma-3-1B-IT	1B	32,768	General
Gemma-3-4B-IT	4B	131,072	General
Mistral-7B-Instruct-v0.2	7B	32,768	General
Qwen2.5-0.5B-Instruct	0.5B	32,768	General
Qwen2.5-1.5B-Instruct	1.5B	32,768	General
Qwen2.5-7B-Instruct	7B	32,768	General
Qwen2.5-Coder-0.5B-Instruct	0.5B	32,768	Code
Qwen2.5-Coder-1.5B-Instruct	1.5B	32,768	Code
Qwen2.5-Coder-3B-Instruct	3B	32,768	Code
Qwen2.5-Coder-7B-Instruct	7B	32,768	Code
Qwen2.5-Math-1.5B-Instruct	1.5B	4,096	Mathematics
Qwen2.5-Math-7B-Instruct	7B	4,096	Mathematics

B.3 Datasets

We evaluate the proposed routing framework on five representative benchmarks spanning science reasoning, mathematical reasoning, code generation, multi-task knowledge understanding, and open-book question answering:

- **ARC-Challenge** (Clark et al., 2018) is a grade-school science question-answering benchmark whose Challenge Set contains questions that are difficult for retrieval-based and word-co-occurrence baselines.
- **GSM8K** (Cobbe et al., 2021) is a dataset of mathematical word problems used to evaluate multi-step mathematical reasoning, problem comprehension, and the generation of a correct calculation process.
- **MBPP** (Austin et al., 2021) is a Python code-generation dataset that requires models to produce executable programs from natural-language problem descriptions, providing a measure of basic program-synthesis ability.
- **MMLU** (Hendrycks et al., 2021) is a large-scale multi-task knowledge-understanding benchmark spanning the humanities, social sciences, natural sciences, engineering, medicine, and other disciplines. It evaluates broad knowledge understanding and problem-solving ability.
- **OpenBookQA** (Mihaylov et al., 2018) is an open-book science question-answering dataset that requires models to combine provided scientific facts with external commonsense knowledge. It evaluates factual knowledge use and compositional reasoning.

B.4 Single-Model Performance

Table 3 reports each checkpoint on the 2,920-query validation split. Cost follows Equation 16; utility is computed as $\mathrm{Accuracy} - 0.5\,\mathrm{Cost}$ to match the main experiments.

Table 3: Single-model performance, sorted by accuracy; utility uses $\alpha = 0.5$.

Checkpoint	Accuracy ↑	Cost ↓	Utility ↑
Gemma-2-9B-IT	**0.8202**	0.2870	**0.6767**
Qwen2.5-Coder-7B-Instruct	0.7445	0.1963	0.6464
Qwen2.5-7B-Instruct	0.7247	0.2707	0.5894
Gemma-3-4B-IT	0.7223	0.4850	0.4798
Qwen2.5-Coder-3B-Instruct	0.6394	0.1622	0.5583
Qwen2.5-1.5B-Instruct	0.6192	0.1398	0.5493
Qwen2.5-Math-7B-Instruct	0.6192	0.7680	0.2352
Qwen2.5-Math-1.5B-Instruct	0.5130	0.7763	0.1249
Qwen2.5-Coder-1.5B-Instruct	0.4955	0.5408	0.2251
Gemma-3-1B-IT	0.4685	0.3825	0.2773
Mistral-7B-Instruct-v0.2	0.4003	0.3347	0.2330
Qwen2.5-0.5B-Instruct	0.3586	0.5776	0.0698
Qwen2.5-Coder-0.5B-Instruct	0.2805	0.6106	-0.0248

B.5 Dataset-Wise Model Heterogeneity

Table 4 shows that the highest-accuracy checkpoint varies by task domain. Specialized Qwen2.5 checkpoints lead on GSM8K and MBPP, while Gemma-2-9B-IT leads on ARC-Challenge, MMLU, and OpenBookQA.

Table 4: Best single model per dataset; cost is normalized response-token cost.

Dataset	Best-accuracy checkpoint	Accuracy ↑	Cost ↓
ARC-Challenge	Gemma-2-9B-IT	0.9197	0.3011
GSM8K	Qwen2.5-Math-1.5B-Instruct	0.9500	0.5703
MBPP	Qwen2.5-Coder-7B-Instruct	0.7444	0.1235
MMLU	Gemma-2-9B-IT	0.7642	0.3270
OpenBookQA	Gemma-2-9B-IT	0.8860	0.2130

Experiments were run on a server with eight NVIDIA RTX A6000 GPUs (48 GB each) and two Intel Xeon Platinum 8336C CPUs (128 logical CPU cores). LLM inference used one model per

GPU process with parallel scheduling across idle GPUs; lightweight router training used automatic device selection.

B.6 Five-Fold Cross-Validation of Candidate Construction

We additionally evaluate the first-stage candidate constructor across five data folds. Table 5 reports candidate hit rate and mean candidate-set size for each held-out fold. Candidate hit remains an upper-bound diagnostic: it measures whether the set retains at least one correct response, not whether downstream aggregation produces the correct final answer.

Table 5: Five-fold candidate-set analysis. The final row reports the mean and standard deviation across folds.

Fold	Candidate hit rate ↑	Mean candidate-set size ↓
CV Fold 0	0.9521	4.1434
CV Fold 1	0.9635	4.9142
CV Fold 2	0.9617	4.9591
CV Fold 3	0.9556	4.9244
CV Fold 4	0.9648	4.9428
Mean $\pm$ std	0.9595 ± 0.0049	4.7768 ± 0.3171

The small fold-to-fold variation indicates stable candidate coverage at this operating point. This analysis is descriptive and does not replace a consolidated held-out evaluation of final-answer accuracy and cost.