

When Routing Collapses: On the Degenerate Convergence of LLM Routers

Guannan Lai^{1,2} Han-Jia Ye^{1,2}

Abstract

LLM routing aims to achieve a favorable quality–cost trade-off by dynamically assigning easy queries to smaller models and harder queries to stronger ones. However, across both unimodal and multimodal settings, we uncover a pervasive yet underexplored failure mode in existing routers: as the user’s cost budget increases, routers systematically default to the most capable and most expensive model even when cheaper models already suffice. As a result, current routers under-utilize small models, wasting computation and monetary cost and undermining the core promise of routing; we term this phenomenon **routing collapse**. We attribute routing collapse to an objective–decision mismatch: many routers are trained to predict scalar performance scores, whereas routing decisions ultimately depend on discrete comparisons among candidate models. Consequently, small prediction errors can flip relative orderings and trigger suboptimal selections. To bridge this gap, we propose **EquiRouter**, a decision-aware router that directly learns model rankings, restoring the role of smaller models and mitigating routing collapse. On RouterBench, EquiRouter reduces cost by about 17% at GPT-4-level performance compared to the strongest prior router. Our code is available at <https://github.com/AIGNLAI/EquiRouter>.

1. Introduction

Large language models (LLMs) deliver state-of-the-art performance across many NLP tasks, but their substantial inference cost hinders scalable deployment (Bang, 2023; Chen & Varoquaux, 2024). A widely adopted mitigation is *LLM routing*, where a lightweight router assigns each incoming query to an appropriate model from a heterogeneous pool,

¹School of Artificial Intelligence, Nanjing University, China

²National Key Laboratory for Novel Software Technology, Nanjing University, China. Correspondence to: Han-Jia Ye <yehj@lamda.nju.edu.cn>.

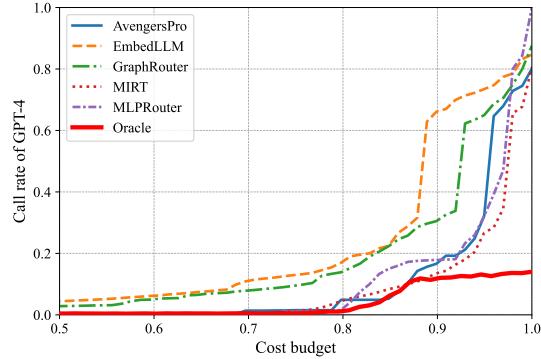


Figure 1. Visualization of **routing collapse** on RouterBench (Hu et al., 2024): under existing routers, as the cost budget increases, the call rate of the strongest model (GPT-4) rapidly saturates near 100%, indicating that routers nearly always select the largest model and largely ignore cheaper alternatives.

sending easy queries to small, inexpensive models and hard queries to larger, more capable ones (Aggarwal et al., 2024; Zhao et al., 2024). Most existing approaches rely on learning, for each query, a predicted performance score for each candidate model, which is then used to make cost-aware selection decisions, and have shown practical benefits in cost-sensitive services (Feng et al., 2025; Ding et al., 2025).

However, we find that existing routers suffer from a widespread and previously underexplored failure mode: as the user budget increases, they systematically select the most powerful and most expensive model, even when cheaper models are already **sufficient**. As shown in Fig. 1, RouterBench spans 11 models with diverse sizes and costs, yet as the cost budget grows, existing routers almost exclusively route queries to GPT-4, while the Oracle uses the strongest model for fewer than 20% of queries on the same benchmark. This indicates that current routers underutilize small models, leading to wasted computation and monetary costs and failing to realize routing’s core benefit: cost reduction. We observe this pattern consistently on both unimodal and multimodal benchmarks, and refer to it as **routing collapse**; this deficiency in leveraging smaller models may also constitute a key bottleneck for effective cost control.

To understand why routing collapse happens, we revisit the problem from a learning perspective. A natural hypothesis is

that collapse reflects poor **generalization**, since routers are trained on one set of queries but deployed on unseen ones. However, even when we train and test on the same data, collapse still persists: routers continue to over-select the strongest model across a wide range of budgets, suggesting that distribution shift is not a necessary condition.

We then examine the routing decision mechanism more directly. Routing selects the best option among all feasible candidate models, so the decision depends on their relative ranking. We find that for the vast majority of queries, multiple models are often closely matched, making the choice inherently fragile: even small perturbations can change which model appears best. Injecting unbiased noise into the Oracle’s per-model performance labels confirms this: as the noise increases, routing quality degrades rapidly. This controlled degradation supports a simple explanation for collapse—*decision sensitivity* in a small-margin regime.

Motivated by this diagnosis, we propose **EquiRouter**, a routing framework designed to consider all candidate models fairly rather than degenerating to a single dominant model. EquiRouter predicts per-instance model rankings instead of scalar scores, which reduces decision flips caused by small prediction errors. To further improve performance, EquiRouter incorporates lightweight model representations to better capture instance-specific differences while preserving full parameter sharing and inference efficiency, making it practical for large-scale deployment.

Since existing evaluation protocols do not explicitly reveal whether a router collapses to overusing the strongest model, we further propose the *Routing Collapse Index (RCI)*, which measures the fraction of queries where the router makes a dominated choice or misses strictly cheaper models that achieve comparable performance. We evaluate EquiRouter on RouterBench and MMR-Bench. Empirically, EquiRouter achieves lower RCI and substantially higher usage of low-cost models, leading to significant cost savings of 17% on RouterBench and 12% on MMR-Bench. Ablation studies further confirm the effectiveness of our key components.

The main contributions of this work are summarized as:

- We conduct a comprehensive empirical study that identifies and characterizes *routing collapse*, a systematic and previously overlooked failure mode of LLM routers.
- We show that routing collapse is primarily caused by a mismatch between the router’s training objective and the deployment-time routing decision objective.
- We introduce **EquiRouter**, a simple yet decision-aware routing framework that directly supervises instance-wise model rankings, consistently achieving significantly improved accuracy–cost trade-offs.
- We propose a new evaluation metric, the *Routing Collapse*

Index (RCI), to quantify the degree of collapse in routers, addressing a gap in existing evaluation protocols.

2. Related Work

Multi-LLM routing has recently attracted growing attention as a general framework for reducing inference cost while maintaining high task performance (Varangot-Reille et al., 2025; Srivatsa et al., 2024). Most existing approaches train a router to predict model performance, utility, or routing decisions directly from the input, and have demonstrated practical gains across a variety of benchmarks and deployment scenarios (Ong et al., 2025; Shen et al., 2023).

Heuristic-based routing Early LLM routing methods largely focus on two-model settings and rely on hand-crafted signals to decide whether to query a strong or a weak model. Representative approaches use confidence estimation, where uncertainty measures or benchmark-derived rules are used to judge whether a model’s response is reliable for a given input (Zhang et al., 2025a; Shnitzer et al., 2023). These heuristics provide simple and lightweight routing policies, but they are often limited in flexibility and do not naturally extend to more diverse model pools.

Learning-based routing A substantial body of work treats routing as a supervised learning problem. In the two-model regime, several methods formulate routing as performance prediction, where the router estimates the expected performance of each model on a query and then makes a cost-aware selection (Ong et al., 2025; Ding et al., 2024).

This paradigm has been extended to multi-model settings by training lightweight routers on query features. For example, *TensorOpera* (Stripelis et al., 2024) and *Triage* (Hari & Thomson, 2023) adopt kNN- and MLP-based routers, while *MixLLM* (Wang et al., 2025) uses linear regression. Other approaches learn query representations and perform clustering to enable model selection, such as K-means on correctness-based representations (Jitkrittum et al., 2025; Zhang et al., 2025b), or use more structured predictors like graph neural networks, as in *GraphRouter* (Feng et al., 2025). Beyond standard supervised routing, *RouterDC* (Chen et al., 2024) applies dual contrastive learning to align the input–output behaviors of different models into a shared space, enabling query-based model composition. Several works also use open-source LLMs as routers to directly make routing decisions (Mohammadshahi et al., 2024; Zhang et al., 2025d; Wang et al., 2026). Finally, *Causal-Router* (Tsiourvas et al., 2025) studies routing from a causal perspective and proposes selecting models by estimating their causal effect on task performance.

Routing Benchmarks To facilitate the study of LLM routing, several benchmarks have been proposed. For uni-

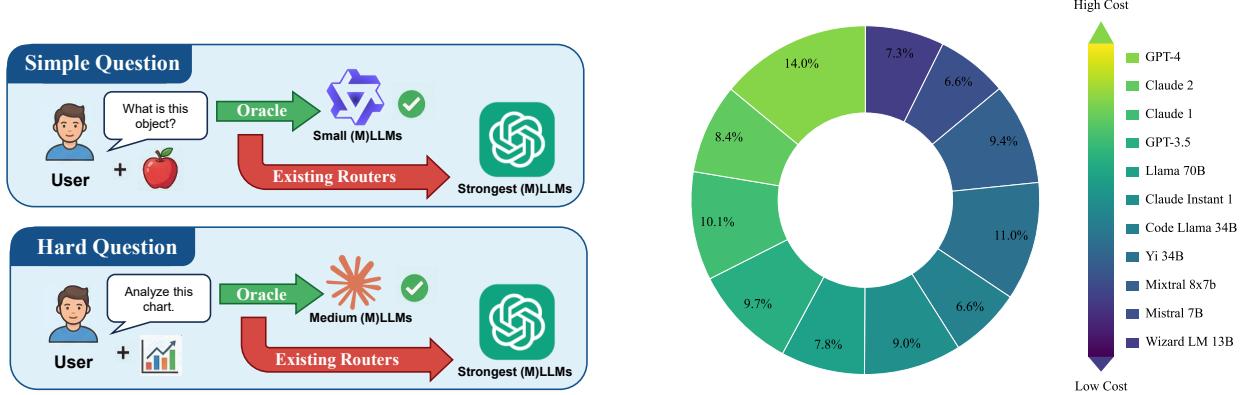


Figure 2. **Left:** An illustration of **routing collapse**: existing routers over-select the strongest model even when smaller models are sufficient. **Right:** Per-model call frequency of the Oracle router on RouterBench, where lighter colors indicate lower-cost models and darker colors indicate higher-cost models.

modal settings, RouterBench (Hu et al., 2024) is the most widely used benchmark and has established a standard experimental protocol for evaluating routing methods, while RouterEval (Huang et al., 2025) provides a more comprehensive and systematic evaluation suite. More recently, MMR-Bench (Ma et al., 2026) extends the routing problem to multimodal scenarios by incorporating both vision and language tasks. *In this work, our empirical analysis is primarily conducted on RouterBench and MMR-Bench.*

3. Routing Collapse: Characterization and Diagnosis

In this section, we define *routing collapse*, the phenomenon in which routers exhibit degenerate convergence to the strongest model as the cost budget increases, and we analyze its causes through both empirical experiments and theoretical investigation.

3.1. Problem Definition

We first give a formal definition of the LLM routing problem. Let the pool of candidate models be denoted by $\mathcal{M} = \{1, \dots, K\}$, where each index $j \in \mathcal{M}$ corresponds to a model. These models may be heterogeneous in capacity, architecture, or inference strategy. Let the query space be $q \in \mathcal{Q}$. For each query q and each model j , let the ground-truth performance be $a_j(q) \in \mathbb{R}$, and let the query-dependent inference cost be $c_j(q) \in \mathbb{R}_+$.

Each user query arrives with an individual cost budget C . For a given query q , the budget-feasible set is

$$\mathcal{F}(q; C) = \{j \in \mathcal{M} \mid c_j(q) \leq C\}. \quad (1)$$

We seek a routing policy $\pi_C : \mathcal{Q} \rightarrow \mathcal{M}$ that maximizes

expected performance:

$$\begin{aligned} & \max_{\pi_C} \mathbb{E}_{q \sim \mathcal{D}} [a_{\pi_C(q)}(q)] \\ \text{s.t. } & \pi_C(q) \in \mathcal{F}(q; C), \forall q. \end{aligned} \quad (2)$$

Oracle Router Given (q, C) , the oracle choice is defined by a lexicographic criterion: it selects a model with the highest performance among feasible candidates, and breaks ties by choosing the lowest cost:

$$j^*(q; C) = \arg \min_{j \in \arg \max_{k \in \mathcal{F}(q; C)} a_k(q)} c_j(q). \quad (3)$$

In practice, however, $a_j(q)$ and $c_j(q)$ are unknown at decision time and can only be obtained by running model j and evaluating its output, which is prohibitively expensive for every query and model. These constraints motivate *learning-based* routers that approximate the oracle using offline supervision.

Existing Learning-based Router A learning-based router uses a parametric predictor to approximate the oracle decision. It is trained on labeled queries $\{(q_n, \mathbf{a}_n, \mathbf{c}_n)\}_{n=1}^N$, where $\mathbf{a}_n = \{a_j(q_n)\}_{j \in \mathcal{M}}$ and $\mathbf{c}_n = \{c_j(q_n)\}_{j \in \mathcal{M}}$. At test time, given $q \sim \mathcal{D}_{\text{test}}$ and budget C , the router predicts $\hat{\mathbf{a}}(q)$ and $\hat{\mathbf{c}}(q)$, restricts to models with $\hat{c}_j(q) \leq C$, and applies the same selection rule as in Eq. (3) with $(\hat{\mathbf{a}}, \hat{\mathbf{c}})$.

3.2. Degenerate Convergence in LLM Routing

We find a pervasive and previously underexplored flaw in existing routers. As illustrated in Fig. 1, when the budget C in Eq. 2 is small, the strongest model is often too expensive, so routers select among cheaper models. As C increases, however, the call frequency of the strongest model rises and eventually converges to nearly 100%. In contrast, on the

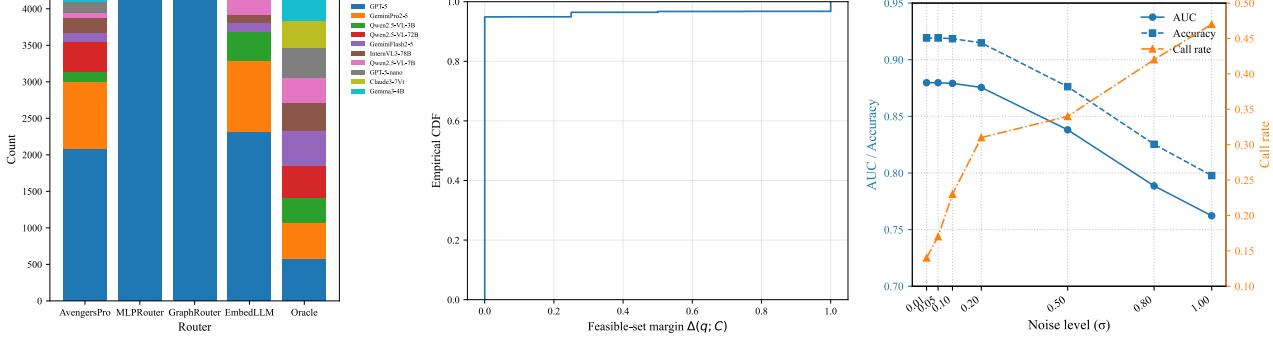


Figure 3. **Left:** Routing collapse observed on MMR-Bench. **Middle:** Feasible-set margin distribution $\Delta(q; C)$ on RouterBench under a large budget, showing that small margins (near-ties) are prevalent. **Right:** Impact of additive Gaussian noise on Oracle per-model performance labels.

same benchmark, the Oracle uses the strongest model for fewer than 20% of queries.

We refer to this failure mode as *routing collapse*. As shown in the left panel of Figure 2, many queries can in fact be easily solved by small or medium-capacity models, yet existing routers still route these queries to the strongest model. This behavior indicates a systematic failure to exploit smaller and cheaper models, which directly contradicts the original motivation of routing. To highlight that this behavior is not inherent to the benchmark, we further compare against the Oracle router. The right panel of Figure 2 shows the idealized per-model call frequency under perfect knowledge of per-query outcomes. Even under loose budget constraints, the Oracle does not concentrate all calls on the most expensive model (GPT-4), and the cheapest model (WizardLM-13B) still receives a non-negligible fraction of queries. This gap between practical routers and the Oracle indicates that routing collapse reflects suboptimal decisions relative to the ground truth, leaving substantial room for improvement.

A similar phenomenon can also be observed on the multimodal benchmark MMR-Bench, as shown in Figure 3 (left). MLPRouter and GraphRouter almost completely degenerate to always invoking the single strongest model. EmbedLLM and AvengersPro exhibit slightly less severe collapse, but still assign more than 80% of the queries to the two strongest models (GPT-5 and GeminiPro2.5), while the remaining smaller models are largely ignored. These results indicate that routing collapse is pervasive across both unimodal and multimodal settings, rather than an artifact of a specific benchmark or modality, and it has become a key bottleneck for cost-effective routing. We next investigate the underlying causes of this failure mode.

3.3. Diagnosing Routing Collapse: From Generalization to Decision Sensitivity

A natural learning-based hypothesis is that routing collapse arises from out-of-sample generalization errors: routers are trained on queries $q \sim \mathcal{D}_{\text{train}}$ but must operate on unseen queries $q \sim \mathcal{D}_{\text{test}}$, and misgeneralization could amplify suboptimal choices under budget constraints.

Generalization is not a necessary condition To test whether distribution shift is required for collapse, we consider a *training-set evaluation* protocol where routers are trained and evaluated on the same dataset, eliminating any train–test generalization gap and providing the most favorable in-sample setting for a learned router. Even under this in-sample evaluation, we observe the same qualitative collapse behavior: learned routers still tend to concentrate their selections on the strongest model and largely ignore the remaining candidates over a wide range of budgets. This indicates that routing collapse does not rely on held-out generalization errors and is unlikely to be explained solely by train–test shift. Implementation details and the full protocol are provided in Appendix B.1.

The above observation motivates a decision-level explanation. Under a per-query budget C , routing selects the top model *within* the budget-feasible set $\mathcal{F}(q; C) = \{j \in \mathcal{M} : c_j(q) \leq C\}$, so the outcome depends primarily on the *relative ordering* of models in $\mathcal{F}(q; C)$, rather than on the absolute calibration of predicted scores. For a given (q, C) , let $a_{(1)}(q; C)$ and $a_{(2)}(q; C)$ denote the best and second-best ground-truth performances among models in $\mathcal{F}(q; C)$, and define the margin

$$\Delta(q; C) = a_{(1)}(q; C) - a_{(2)}(q; C). \quad (4)$$

When $\Delta(q; C)$ is small, even modest prediction errors can flip the top-1 ordering within $\mathcal{F}(q; C)$, causing decision errors even under small prediction errors and revealing an objective–decision mismatch between pointwise score pre-

diction and deployment-time arg max selection.

Small margins are prevalent in RouterBench We next examine the feasible-set margin distribution on RouterBench. For the largest cost budget (i.e., C sufficiently large such that $\mathcal{F}(q; C) = \mathcal{M}$ for almost all queries), the empirical margin distribution is highly concentrated at zero: $\Pr(\Delta(q; C) \leq \epsilon) = 94.90\%$ for $\epsilon \in \{0, 10^{-3}, 10^{-2}, 5 \times 10^{-2}\}$, with a tie rate $\Pr(\Delta(q; C) \approx 0) = 94.90\%$ (Fig. 3, middle). This indicates that RouterBench operates predominantly in a *small-margin* regime, where the best and second-best models within the feasible set are often nearly indistinguishable in ground-truth performance.

Why small margins lead to collapse Routing makes an arg max decision within $\mathcal{F}(q; C)$, so it depends only on the *ordering* of predicted scores. When the top candidates are nearly tied, even small prediction errors can flip this ordering. For example, if two feasible models have $a_{j_1}(q) = 0.801$ and $a_{j_2}(q) = 0.800$ (so $\Delta(q; C) = 10^{-3}$), then a perturbation of size 10^{-3} is enough to change which model is ranked first and therefore selected. When such near-ties dominate (as in RouterBench; Fig. 3, middle), these top-1 flips become frequent across queries. As the budget increases and stronger models become feasible more often, this instability systematically concentrates selections on the strongest model, manifesting as routing collapse.

To demonstrate this sensitivity in a controlled way, we inject unbiased noise into the oracle per-model performance labels. Concretely, for each query q and model $j \in \mathcal{M}$, we define

$$\hat{a}_j(q) = a_j(q) + \varepsilon_j(q), \quad \varepsilon_j(q) \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma^2),$$

and vary the noise scale σ .

Fig. 3 (right) shows that as σ increases, the oracle router gradually exhibits *routing collapse*, with selections increasingly concentrating on the strongest model. This confirms that routing is fundamentally comparison-based: decisions are made by ranking candidate models within the budget-feasible set, so even small perturbations can swap the relative order of near-tied models and change the selected model. More broadly, the result highlights an objective–decision mismatch: learning to predict accurate scalar scores does not necessarily translate into correct discrete rankings, and small residual errors can be amplified into systematic shifts in selection behavior, including an over-selection of the strongest model.

Takeaway Overall, routing collapse is driven by rank instability in a small-margin regime: routing relies on within-feasible-set comparisons, and small score perturbations can flip the selected model as the feasible set expands with C . This reveals an objective–decision mismatch, suggesting that avoiding collapse requires learning objectives that directly align with the deployment-time ranking decision.

4. EquiRouter: Paying Attention to All Models

Based on the analysis in Section 3, routing decisions are inherently comparison-based and can be brittle when learned routers predict scalar scores that are later consumed by discrete model comparisons. We therefore seek a routing design whose training objective directly matches this comparison structure. To this end, we propose **EquiRouter**, which adopts a ranking-first objective: instead of predicting scalar scores and then relying on a brittle downstream arg max, it directly supervises the per-query *ordering* of all candidate models. Crucially, each query induces comparison signals that involve *every* model (via pairwise/listwise ranking), preventing smaller models from being ignored during training and improving robustness in the small-margin regime that triggers collapse.

Overview. As shown in Fig. 4, given an encoded query vector $\mathbf{q} \in \mathbb{R}^{d_q}$ and a set of K learnable model embeddings $\{m_j\}_{j=1}^K$, EquiRouter first extracts shared query features with a common trunk and then constructs, for each candidate model, a lightweight query-dependent, model-specific representation to capture instance-level differences. The router then outputs comparison scores for all models and is trained with a ranking-aware loss $\mathcal{L}_{\text{rank}}$ so that the predicted ordering matches the ground-truth ordering. At inference time, given a budget C , the router filters out models with $\hat{c}_j(q) > C$ and selects the highest-ranked model from the remaining feasible set.

4.1. Query and Model Representations

We maintain a learnable embedding $m_j \in \mathbb{R}^d$ for each model $j \in \{1, \dots, K\}$ in the model pool. These model embeddings are randomly initialized and trained jointly with the router. Intuitively, they serve as compact, trainable descriptors of the relative behavior and capability profiles of different models, and provide a unified interface for comparing heterogeneous models within the routing network.

For each query $q \in \mathcal{Q}$, we obtain a fixed embedding using a pretrained encoder:

$$\mathbf{q} = \text{Enc}(q), \quad \mathbf{q} \in \mathbb{R}^{d_q}.$$

where $\text{Enc}(\cdot)$ can be instantiated by either a lightweight sentence encoder (e.g., all-MiniLM-L6-v2 ([Reimers & Gurevych, 2019](#))) or a stronger embedding model (e.g., Qwen3-Embedding-0.6B ([Zhang et al., 2025c](#))), depending on the deployment setting. In all cases, the encoder is kept fixed and only serves to provide a semantic representation of the input query. These query and model representations constitute the basic inputs to EquiRouter, upon which subsequent model-conditioned modulation and ranking-based comparison are performed.

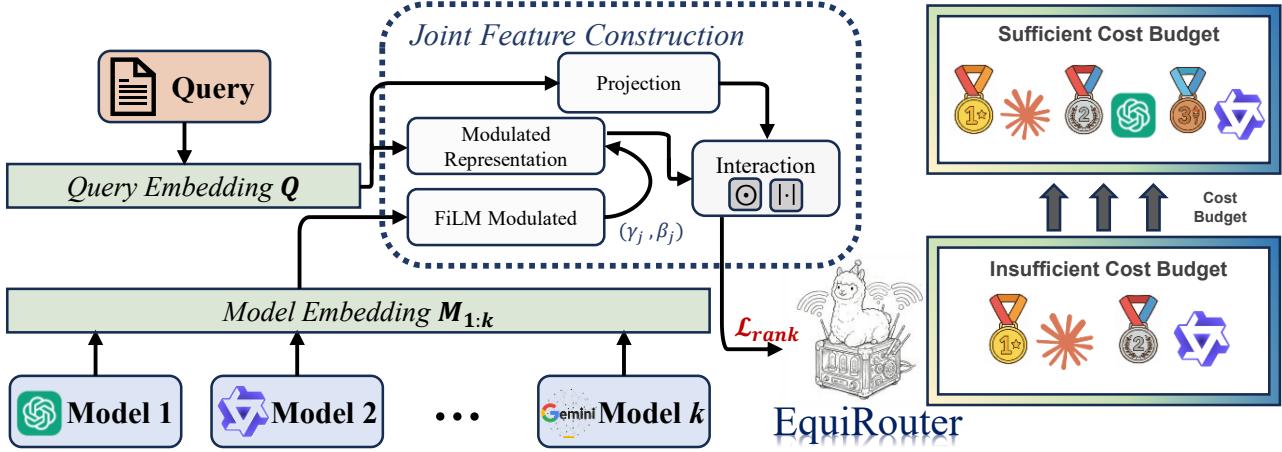


Figure 4. Overview of EquiRouter: given a query and a set of candidate models, the router constructs query-conditioned model representations and predicts their per-query ranking with a decision-aligned, ranking-aware objective.

4.2. Model-Conditioned Model Representations

Motivation To make routing decisions sensitive to per-query differences between models, it is important to obtain representations that are both model-specific and conditioned on the current query. Static model descriptors risk losing the instance-level contrasts that determine which model is preferable for a given input; to recover these contrasts, we construct *model-conditioned* query features, so that the same query can be viewed through model-specific lenses that highlight instance-level contrasts among candidate models.

Let the query embedding be $\mathbf{q} \in \mathbb{R}^{d_q}$. The router maintains a learned embedding $m_j \in \mathbb{R}^{d_m}$ for each model $j \in \{1, \dots, K\}$. A shared trunk

$$f : \mathbb{R}^{d_q} \rightarrow \mathbb{R}^D, \quad q \mapsto z$$

maps the query embedding to a latent vector $z \in \mathbb{R}^D$. Subsequent operations are applied per model j to produce a scalar compatibility score $s_j(q)$.

Modulated representation To obtain model-specific query features, we adopt a FiLM-style affine modulation (Perez et al., 2018) parameterized by the model embedding. Compared to using a static model descriptor, FiLM provides a lightweight way to condition a shared trunk on each candidate model: it rescales and shifts the shared query features in a model-dependent manner, allowing the router to view the same query through different “model lenses” and thus better capture per-query, model-specific contrasts that matter for ranking. Importantly, this conditioning adds only $O(D)$ parameters per model embedding through two linear projections and incurs negligible compute overhead, making it suitable for routing over large model pools. Let

$$\phi : \mathbb{R}^{d_m} \rightarrow \mathbb{R}^{2D}, \quad \psi : \mathbb{R}^{d_m} \rightarrow \mathbb{R}^D$$

be learned linear projections, and write $\phi(m_j) = [\gamma_j; \beta_j]$ with $\gamma_j, \beta_j \in \mathbb{R}^D$. The resulting model-conditioned query feature is

$$z_j = \gamma_j \odot z + \beta_j, \quad (5)$$

and the projected model vector is

$$e_j = \psi(m_j) \in \mathbb{R}^D. \quad (6)$$

Both ϕ and ψ are implemented as small linear layers.

Joint feature To capture complementary interaction patterns between the conditioned query z_j and the projected model vector e_j , we form a compact joint feature

$$h_j = [z_j, e_j, z_j \odot e_j, |z_j - e_j|] \in \mathbb{R}^{4D}. \quad (7)$$

Here z_j and e_j provide conditioned baselines, $z_j \odot e_j$ captures multiplicative compatibility and feature-wise importance, and $|z_j - e_j|$ encodes relative offsets and asymmetries. The resulting vector h_j is then passed to the downstream comparison/scoring head to produce the scalar score $s_j(q)$.

4.3. Scoring and Training Objective

After constructing joint features $h_j(q)$ for every query–model pair, the router produces a scalar score for each candidate model via a shared comparison head. Concretely we write

$$s_j(q) = g_\theta(h_j(q)),$$

where $g_\theta : \mathbb{R}^{4D} \rightarrow \mathbb{R}$ denotes a small, parameterized scoring function. Collecting scores into a vector $\mathbf{s}(q) = [s_1(q), \dots, s_K(q)]^\top \in \mathbb{R}^K$ defines the router’s predicted ordering.

Ideally, these scores need not recover absolute performance values; they only need to preserve the correct per-query ordering used by routing. To enforce this, we supervise $\mathbf{s}(q)$ with a pairwise ranking loss. Let $\mathbf{a}(q) =$

$[a_1(q), \dots, a_K(q)]^\top \in \mathbb{R}^K$ denote the ground-truth performance vector and $\mathbf{c}(q) \in \mathbb{R}_+^K$ the corresponding per-model costs for query q . Since routing prioritizes higher performance and, when performance ties, lower cost, we define the set of ordered pairs

$$\mathcal{P}(q) = \left\{ (i, j) \mid \begin{array}{l} a_i(q) > a_j(q) \text{ or} \\ (a_i(q) = a_j(q) \wedge c_i(q) < c_j(q)) \end{array} \right\}.$$

We use the logistic pairwise loss

$$\mathcal{L}_{\text{rank}} = \frac{1}{|\mathcal{P}(q)|} \sum_{(i,j) \in \mathcal{P}(q)} \log \left(1 + \exp \left(-(s_i(q) - s_j(q)) \right) \right). \quad (8)$$

The full training objective minimizes the expected ranking loss over the data distribution plus standard regularization:

$$\mathcal{L}(\Theta) = \mathbb{E}_{q \sim \mathcal{D}} [\mathcal{L}_{\text{rank}}(\mathbf{s}(q), \mathbf{a}(q))] + \lambda \mathcal{R}(\Theta),$$

where \mathcal{R} is a generic weight-regularizer (e.g. ℓ_2 penalty) and λ is a tunable coefficient. The parameter set Θ trained end-to-end includes the router components:

$$\Theta = \{f, \phi, \psi, g_\theta, \{m_j\}_{j=1}^K\},$$

where m_j are the model embeddings, f is the shared query trunk, ϕ, ψ are the lightweight projection layers used for modulation and projection, and g_θ is the scoring head.

5. Experiment

5.1. Experimental Setup

We conduct experiments on both the unimodal RouterBench (Hu et al., 2024) benchmark and the multimodal MMR-Bench (Ma et al., 2026) benchmark to further investigate the routing collapse phenomenon and to evaluate the effectiveness of our proposed EquiRouter. In this section, we describe the datasets and data splits, the evaluation protocols, and the baseline methods used for comparison.

Benchmarks Splits For each benchmark, we split the data into training, validation, and test sets with a ratio of 3:1:6. This split is intended to reflect a practical setting in which constructing routing supervision is relatively costly: the training set is kept deliberately small to control data collection overhead, while still being sufficient to learn an effective router.

Evaluation Metrics Following standard practice in routing and model selection (Jitkrittum et al., 2025; Zhuang et al., 2024), we evaluate routing performance by tracing a performance–cost curve obtained by varying the user cost budget. Prior work typically summarizes this curve with three metrics: **(1) nAUC** (Normalized AUC), the normalized area under the performance–cost curve measuring overall

efficiency across budgets; **(2) P_s** (Peak Score), the best performance achieved on the curve together with its corresponding cost; and **(3) QNC** (Quality–Neutral Cost), the relative cost required to match the performance of the most accurate standalone model.

However, these curve-level metrics cannot reveal whether a router makes *dominated* decisions on individual queries, and thus may fail to detect routing collapse, since a router can appear strong in terms of accuracy or AUC simply by frequently selecting the most capable model even when cheaper models perform better on the same query.

To explicitly quantify this failure mode, we define the *Routing Collapse Index (RCI)*, which captures both outright dominated choices and missed opportunities to use strictly cheaper models when performance is equivalent. Let N be the number of evaluation queries, m_n the model selected for query n , $a_{n,j}$ the observed performance of model j on query n , and $c_{n,j}$ the per-query cost of model j on query n . For each query n , define the best-achievable performance $a_n^* = \max_j a_{n,j}$, and let $S_n = \{j : c_{n,j} < c_{n,m_n}\}$ denote the set of models strictly cheaper than the selected model on the same query, with $X_n := |S_n|$.

We assign a per-query collapse score s_n as follows:

$$s_n = \begin{cases} 1, & a_{n,m_n} < a_n^*, \\ \frac{K_n}{X_n}, & a_{n,m_n} = a_n^* \text{ and } X_n > 0, \\ 0, & a_{n,m_n} = a_n^* \text{ and } X_n = 0, \end{cases}$$

where

$$K_n = |\{j \in S_n : a_{n,j} \geq a_{n,m_n}\}|$$

is the number of strictly cheaper models that achieve at least the selected model’s performance on query n . Intuitively, $s_n = 1$ flags queries where the router failed to pick any optimal model; when the router picks an optimal model, s_n measures the fraction of strictly cheaper alternatives that would have matched that optimal performance.

The Routing Collapse Index is the mean per-query score:

$$\text{RCI} = \frac{1}{N} \sum_{n=1}^N s_n \in [0, 1].$$

Comparison Methods We compare our approach against a broad set of representative routing methods, including AvengersPro (Zhang et al., 2025b), EmbedLLM (Zhuang et al., 2024), GraphRouter (Feng et al., 2025), ML-PRouter and kNNRouter (Stripelis et al., 2024), CausalRouter (Tsiourvas et al., 2025), as well as MIRT and NIRT (Song et al., 2025). These methods cover a wide range of routing paradigms, including embedding-based matching,

Table 1. Comparison of routing methods on RouterBench and MMR-Bench. **Bold numbers** indicate the best results, underlined numbers indicate the second-best results, and / in QNC denotes that the method cannot reach the performance of the strongest model.

Router	Venue	RouterBench				MMR-Bench			
		nAUC (\uparrow)	QNC (\downarrow)	P_s (\uparrow)	RCI (\downarrow)	nAUC (\uparrow)	QNC (\downarrow)	P_s (\uparrow)	RCI (\downarrow)
kNNRouter	[EMNLP '24]	0.7541	<u>0.9381</u>	0.8038	0.7847	0.6959	0.9895	0.7416	0.7895
MLPRouter		0.7448	1.0000	0.8035	0.7871	0.6815	1.0000	0.7384	0.7819
EmbedLLM	[ICLR '24]	0.7427	/	0.8027	0.7869	0.6863	/	0.7369	0.7546
GraphRouter		0.7477	0.9470	0.8038	0.7867	0.6214	1.0000	0.7384	0.7819
MIRT	[ACL '25]	0.7483	/	0.8032	0.7870	0.6904	0.9995	0.7417	0.7828
NIRT		0.7465	0.9987	0.8036	0.7872	0.6974	<u>0.9815</u>	0.7454	0.7840
AvengersPro	[DAI '25]	<u>0.7555</u>	/	0.8033	0.7810	<u>0.7007</u>	/	0.7362	0.7280
CausalRouter	[NeurIPS '25]	0.7055	0.9954	0.8043	<u>0.7571</u>	0.6914	1.0517	0.7442	0.7140
<i>EquiRouter</i>		0.7712	0.7731	<u>0.8041</u>	0.6911	0.7059	0.8784	0.7430	0.6949
<i>w/o joint feature</i>		0.7549	0.9154	0.8037	0.7043	0.7041	0.9395	0.7415	0.7025
<i>w/o ranking loss</i>		0.7623	0.9637	0.8045	0.7325	0.7044	0.9752	0.7437	0.7135

graph-based routing, parametric and non-parametric predictors, causal modeling, and item-response-theory-based approaches, providing a comprehensive set of baselines.

5.2. Experimental Results

Main result Table 1 summarizes the comparative evaluation on RouterBench and MMR-Bench. EquiRouter achieves the highest overall nAUC on both benchmarks while also obtaining the lowest RCI on each dataset. In other words, EquiRouter not only improves the accuracy–cost trade-off but also substantially reduces the tendency to select unnecessarily expensive models. Notably, EquiRouter demonstrates a significant improvement in QNC: on RouterBench, it achieves GPT-4-level performance using only 77% of the cost, compared to over 93% for the best previous routers; on MMR-Bench, it achieves GPT-5-level performance using only 87% of the cost, compared to 98% for prior methods. This indicates that EquiRouter effectively leverages smaller models, deploying them more frequently to reduce overall cost without sacrificing performance.

Ablation Results The ablations in Table 1 highlight the roles of our two core components. *w/o joint feature* removes the model-conditioned joint feature construction, while *w/o ranking loss* replaces the ranking loss in Eq. (8) with an MSE objective. Removing the ranking loss substantially weakens cost control and collapse mitigation: on RouterBench, QNC increases from 0.773 to 0.964 and RCI from 0.691 to 0.733; on MMR-Bench, QNC increases from 0.878 to 0.975 and RCI from 0.695 to 0.714. This confirms that the ranking objective is crucial for aligning training with the comparison-based routing decision and avoiding systematic overuse of expensive models.

Removing model-conditioned model representations primarily affects per-query discrimination and overall performance. On RouterBench, nAUC drops from 0.771 to 0.755, QNC

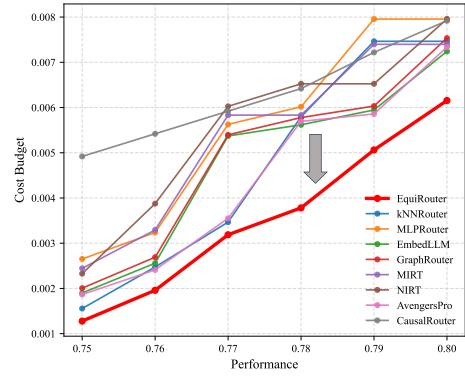


Figure 5. Average cost required to achieve varying performance targets on RouterBench. Lower curves indicate more efficient use of models and better cost-performance trade-offs.

rises from 0.773 to 0.915, and RCI slightly increases from 0.691 to 0.704; similar trends are observed on MMR-Bench. This demonstrates that conditioning model embeddings on the query enhances the router’s ability to identify queries suitable for smaller models, improving the accuracy–cost trade-off and reducing unnecessary high-cost selections.

Significant cost reduction with EquiRouter Figure 5 compares the average cost required by different routing methods to achieve the same performance on RouterBench. EquiRouter consistently maintains the lowest cost across all performance levels, with the relative improvement increasing as performance targets rise. On average, EquiRouter reduces the per-query cost by approximately \$0.002, corresponding to a roughly 17–25% decrease compared to existing methods. This demonstrates that EquiRouter effectively leverages smaller models throughout the routing process, thereby mitigating the occurrence of routing collapse.

6. Conclusion

In this work, we identify and characterize *routing collapse*, a pervasive failure mode in which learned routers increasingly overuse the strongest model as the budget grows, undermining cost-effective routing. We provide empirical diagnostics that rule out poor generalization and show that an objective–decision mismatch is the primary cause. We then propose a decision-aware routing approach and introduce the *Routing Collapse Index* to quantify collapse. Experiments on Router-Bench and MMR-Bench demonstrate a better accuracy–cost trade-off and substantial cost reductions.

References

- Aggarwal, P., Madaan, A., Anand, A., Potharaju, S. P., Mishra, S., Zhou, P., Gupta, A., Rajagopal, D., Kappaganthu, K., Yang, Y., et al. Automix: Automatically mixing language models. In *NeurIPS*, 2024.
- Bang, F. Gptcache: An open-source semantic cache for llm applications enabling faster answers and cost savings. In *NLP-OSS*, 2023.
- Chen, L. and Varoquaux, G. What is the role of small models in the llm era: A survey. In *arXiv preprint arXiv:2409.06857*, 2024.
- Chen, S., Jiang, W., Lin, B., Kwok, J., and Zhang, Y. Routerdc: Query-based router by dual contrastive learning for assembling large language models. In *NeurIPS*, 2024.
- Ding, D., Mallick, A., Wang, C., Sim, R., Mukherjee, S., Rühle, V., Lakshmanan, L. V. S., and Awadallah, A. H. Hybrid LLM: Cost-efficient and quality-aware query routing. In *ICLR*, 2024.
- Ding, D., Mallick, A., Zhang, S., Wang, C., Madrigal, D., Garcia, M. D. C. H., Xia, M., Lakshmanan, L. V., Wu, Q., and Rühle, V. Best-route: Adaptive llm routing with test-time optimal compute. In *ICML*, 2025.
- Feng, T., Shen, Y., and You, J. Graphrouter: A graph-based router for llm selections. In *ICLR*, 2025.
- Hari, S. N. and Thomson, M. Tryage: Real-time, intelligent routing of user prompts to large language models. *arXiv preprint arXiv:2308.11601*, 2023.
- Hu, Q. J., Bieker, J., Li, X., Jiang, N., Keigwin, B., Ranaganath, G., Keutzer, K., and Upadhyay, S. K. Routerbench: A benchmark for multi-llm routing system. In *arXiv preprint arXiv:2403.12031*, 2024.
- Huang, Z., Ling, G., Lin, Y., Chen, Y., Zhong, S., Wu, H., and Lin, L. Routereval: A comprehensive benchmark for routing llms to explore model-level scaling up in llms. In *EMNLP*, 2025.
- Jitkrittum, W., Narasimhan, H., Rawat, A. S., Juneja, J., Wang, Z., Lee, C.-Y., Shenoy, P., Panigrahy, R., Menon, A. K., and Kumar, S. Universal llm routing with correctness-based representation. In *First Workshop on Scalable Optimization for Efficient and Adaptive Foundation Models*, 2025.
- Ma, H., Lai, G., and Ye, H.-J. Mmr-bench: A comprehensive benchmark for multimodal llm routing. *arXiv preprint arXiv:2601.17814*, 2026.
- Mohammadshahi, A., Shaikh, A. R., and Yazdani, M. Routoo: Learning to route to large language models effectively. *arXiv preprint arXiv:2401.13979*, 2024.
- Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., Gonzalez, J. E., Kadous, M. W., and Stoica, I. RouteLLM: Learning to route LLMs from preference data. In *ICLR*, 2025.
- Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.
- Reimers, N. and Gurevych, I. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *EMNLP-IJCNLP*, 2019.
- Shen, Y., Song, K., Tan, X., Li, D., Lu, W., and Zhuang, Y. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. In *NeurIPS*, 2023.
- Shnitzer, T., Ou, A., Silva, M., Soule, K., Sun, Y., Solomon, J., Thompson, N., and Yurochkin, M. Llm routing with benchmark datasets. In *NeurIPS 2023 Workshop on Distribution Shifts: New Frontiers with Foundation Models*, 2023.
- Song, W., Huang, Z., Cheng, C., Gao, W., Xu, B., Zhao, G., Wang, F., and Wu, R. IRT-router: Effective and interpretable multi-LLM routing via item response theory. In *ACL*, 2025.
- Srivatsa, K., Maurya, K. K., and Kochmar, E. Harnessing the power of multiple minds: Lessons learned from llm routing. In *NAACLW*, 2024.
- Stripelis, D., Hu, Z., Zhang, J., Xu, Z., Shah, A. D., Jin, H., Yao, Y., Avestimehr, S., and He, C. Tensoropera router: A multi-model router for efficient llm inference. In *EMNLP*, 2024.
- Tsiourvas, A., Sun, W., and Perakis, G. Causal llm routing: End-to-end regret minimization from observational data. In *NeurIPS*, 2025.

Varangot-Reille, C., Bouvard, C., Gourru, A., Ciancone, M., Schaeffer, M., and Jacquetin, F. Doing more with less—implementing routing strategies in large language model-based systems: An extended survey. *arXiv preprint arXiv:2502.00409*, 2025.

Wang, C., Li, H., Zhang, Y., Chen, L., Chen, J., Jian, P., Ye, P., Zhang, Q., and Hu, S. Icl-router: In-context learned model representations for llm routing. In *AAAI*, 2026.

Wang, X., Liu, Y., Cheng, W., Zhao, X., Chen, Z., Yu, W., Fu, Y., and Chen, H. Mixllm: Dynamic routing in mixed large language models. *arXiv preprint arXiv:2502.18482*, 2025.

Zhang, T., Mehradfar, A., Dimitriadis, D., and Avestimehr, S. Leveraging uncertainty estimation for efficient llm routing. *arXiv preprint arXiv:2502.11021*, 2025a.

Zhang, Y., Li, H., Chen, J., Zhang, H., Ye, P., Bai, L., and Hu, S. Beyond gpt-5: Making llms cheaper and better via performance-efficiency optimized routing. In *DAI*, 2025b.

Zhang, Y., Li, M., Long, D., Zhang, X., Lin, H., Yang, B., Xie, P., Yang, A., Liu, D., Lin, J., et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025c.

Zhang, Y.-K., Zhan, D.-C., and Ye, H.-J. Capability instruction tuning: A new paradigm for dynamic llm routing. In *AAAI*, 2025d.

Zhao, Z., Jin, S., and Mao, Z. M. Eagle: Efficient training-free router for multi-llm inference. *arXiv preprint arXiv:2409.15518*, 2024.

Zhuang, R., Wu, T., Wen, Z., Li, A., Jiao, J., and Ramchandran, K. Embedllm: Learning compact representations of large language models. In *ICLR*, 2024.

Appendix

The appendix is organized as follows:

- **Appendix A (Reproducibility & Protocol).** Dataset splits, cost definition and budget scanning protocol, training details, and baseline implementation settings.
- **Appendix B (Further Analysis of Routing Collapse).** Additional diagnostics that complement Section 3, including the training-set evaluation protocol and results.
- **Appendix C (Cost Prediction).** Cost-prediction setup used by all routers and supporting sanity checks.
- **Appendix D (Routing Collapse Index).** Definition details, tie-handling rules, and boundary cases, with illustrative examples.
- **Appendix E (Supplementary Experimental Results).** Additional plots/analyses (e.g., per-model call rates) that support the main findings.
- **Appendix F (Complexity Analysis).** Parameter and runtime complexity of EquiRouter.

A. Reproducibility & Protocol

A.1. Datasets and Splits

RouterBench. We use RouterBench (Hu et al., 2024), a large-scale benchmark for evaluating budget-aware routing over text-only LLMs. RouterBench aggregates diverse task families, including commonsense reasoning (HellaSwag, WinoGrande, ARC-Challenge), knowledge-intensive QA (MMLU), open-ended chat (MT-Bench), math reasoning (GSM8K), code generation (MBPP), and an RAG subset constructed from real user queries. The benchmark provides per-query outcomes across a fixed pool of candidate models, enabling offline evaluation of routing policies under varying cost budgets.

MMR-Bench. To study multimodal routing, we use MMR-Bench (Ma et al., 2026), which evaluates routing over heterogeneous MLLMs with a unified, budget-aware protocol. It covers three routing-relevant scenarios, including document-centric OCR and understanding (OCRBench, SEED-Bench v2 Plus), general VQA and grounding (MMStar, RealWorldQA), and multimodal math and diagram reasoning (MathVerse, MathVista, MathVision). Similar to RouterBench, MMR-Bench provides precomputed instance–model utilities and normalized costs, allowing reproducible offline comparisons across routers.

Splits and randomness control. For both benchmarks, we create a fixed 3:1:6 split for train/validation/test with random seed 42. We allocate a large test portion to obtain stable estimates of performance–cost frontiers and collapse-related statistics across budgets, while reserving a validation

set for model selection and early stopping. All methods are trained and tuned only on the training and validation splits, and we report results exclusively on the held-out test split.

We define cost as the per-query monetary cost computed from token pricing. Concretely, for each model j and query q , $c_j(q)$ equals the total number of billed tokens multiplied by the model’s unit price. To trace the performance–cost curve, we sweep the budget C over an evenly spaced grid of 100 values between the minimum and maximum feasible costs, yielding a dense approximation of the curve used to compute nAUC, P_s , and QNC.

A.2. Baselines and Hyperparameters

We summarize the compared routers and the hyperparameters used in our implementation. Unless otherwise specified, all learned predictors are trained for 30 epochs with batch size 2048, Adam optimizer, learning rate 10^{-3} . On RouterBench, we use `all-MiniLM-L6-v2` (Reimers & Gurevych, 2019) to encode queries, producing $d_q = 384$ -dimensional embeddings. On MMR-Bench, we use ViTB/16 to encode images and concatenate the visual feature with the text feature to form a $d_q = 1024$ -dimensional multimodal embedding. For methods requiring external textual descriptions of candidate models (e.g., GraphRouter, MIRT, and NIRT), we generate these descriptions using Qwen3-8B.

- **kNNRouter.** A non-parametric router that retrieves nearest neighbors of a query in the embedding space and transfers their observed routing labels to make a decision. We set the number of neighbors to $k = 50$.
- **MLPRouter.** A two-layer MLP that maps the query embedding to per-model performance predictions and then applies the same budget-feasible selection rule as in the main text.
- **GraphRouter.** A graph-based router that models cross-model structure (e.g., dependencies among candidate models) and predicts per-model performance for routing.
- **EmbedLLM.** An embedding-based learned router that predicts per-model performance from query representations, followed by budget-feasible selection at inference.
- **CausalRouter.** A learned router that incorporates causal-style features/regularization when predicting per-model performance, and then selects the best feasible model under the budget.
- **AvengersPro.** A clustering-based router that partitions queries into groups and assigns routing decisions based on cluster membership. We use 200 clusters with `n_init=auto`, `max_iter=1000`, and the `elkan` algorithm for clustering.
- **MIRT.** A multi-dimensional item-response-theory router that represents each query with a latent vector and pre-

dicts per-model performance via a factorized formulation. Model descriptions used by this baseline are generated with Qwen3–8B.

- **NIRT.** A neural item-response-theory router that replaces the linear mapping in IRT with a neural predictor to capture non-linear query–model interactions. We construct external knowledge used by this baseline via the DeepSeek API.

A.3. Evaluation Metrics

Following standard practice in routing and model selection (Jitkrittum et al., 2025; Zhuang et al., 2024), we evaluate routing performance by tracing a performance–cost curve obtained by sweeping the per-query budget C . Let $\pi_C : \mathcal{Q} \rightarrow \mathcal{M}$ denote the routing policy under budget C , and define its test performance and expected cost as

$$A(C) = \mathbb{E}_{q \sim \mathcal{D}_{\text{test}}} [a_{\pi_C(q)}(q)],$$

$$\text{Cost}(C) = \mathbb{E}_{q \sim \mathcal{D}_{\text{test}}} [c_{\pi_C(q)}(q)].$$

We evaluate $\{\text{Cost}(C), A(C)\}$ over a grid of budgets to obtain the performance–cost curve and report three curve-level summary metrics.

- (1) **nAUC.** Let $\mathcal{C} = \{C_1, \dots, C_T\}$ be the set of evaluated budgets ordered by increasing $\text{Cost}(C)$. Define $x_t = \text{Cost}(C_t)$ and $y_t = A(C_t)$. The normalized area under the curve is computed by trapezoidal integration and normalized by the cost range:

$$\text{nAUC} = \frac{1}{x_T - x_1} \sum_{t=1}^{T-1} \frac{(y_t + y_{t+1})}{2} (x_{t+1} - x_t).$$

- (2) **P_s (Peak Score).** We define the peak performance on the curve as

$$P_s = \max_{t \in \{1, \dots, T\}} y_t,$$

and report it together with the corresponding cost x_{t^*} , where $t^* \in \arg \max_t y_t$.

- (3) **QNC (Quality–Neutral Cost).** Let A_{\max} denote the test performance of the single most accurate standalone model in the pool, i.e.,

$$A_{\max} = \max_{j \in \mathcal{M}} \mathbb{E}_{q \sim \mathcal{D}_{\text{test}}} [a_j(q)].$$

We define the quality–neutral cost as the minimal expected cost required by the router to reach this quality:

$$\text{QNC} = \min_{t: y_t \geq A_{\max}} x_t,$$

and report QNC/x_{\max} as a relative cost, where

$$x_{\max} = \mathbb{E}_{q \sim \mathcal{D}_{\text{test}}} [c_{j_{\max}}(q)]$$

$$j_{\max} \in \arg \max_{j \in \mathcal{M}} \mathbb{E}_{q \sim \mathcal{D}_{\text{test}}} [a_j(q)].$$

If the router never achieves A_{\max} on the evaluated curve, we report QNC as “/”. We introduce RCI in Section D.

B. Further Analysis of Routing Collapse

In this appendix, we provide additional analyses that complement the main diagnosis in Section 3. In particular, we describe the *training-set evaluation* protocol in detail and report the corresponding results. This experiment isolates whether routing collapse depends on train–test distribution shift by evaluating routers under the most favorable in-sample condition.

B.1. Experiment 1: Training-set Evaluation

Setup. Let $\mathcal{M} = \{1, \dots, K\}$ denote the model pool. We consider a labeled routing dataset $\{(q_n, \mathbf{a}_n, \mathbf{c}_n)\}_{n=1}^N$, where $\mathbf{a}_n = [a_1(q_n), \dots, a_M(q_n)]^\top$ and $\mathbf{c}_n = [c_1(q_n), \dots, c_M(q_n)]^\top$. For a per-query budget C , the feasible set is $\mathcal{F}(q_n; C) = \{j \in \mathcal{M} : c_j(q_n) \leq C\}$, and the oracle choice follows the rule in Eq. (3).

In training-set evaluation, we train each learned router on the full dataset and then evaluate it on the same set of queries $\{q_n\}_{n=1}^N$. This eliminates any generalization gap between training and testing and provides the most favorable in-sample condition for a learned router. All hyperparameters, early-stopping criteria, and training schedules are kept identical to the standard train/test split used in the main experiments; the only difference is that evaluation is performed on the training queries. For each budget C , the router selects a model $\hat{j}_n(C)$ for each query q_n , and we report nAUC, P_s , as well as *Cheapest Calls* and *Strongest Calls*.

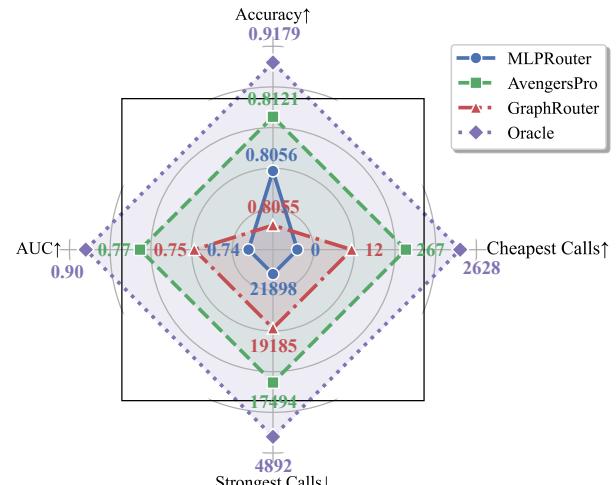


Figure A1. Training-set evaluation on RouterBench. Routers are trained and evaluated on the same dataset to eliminate any train–test generalization gap. We report Accuracy, AUC, and the selection rates of the strongest and cheapest models across budgets.

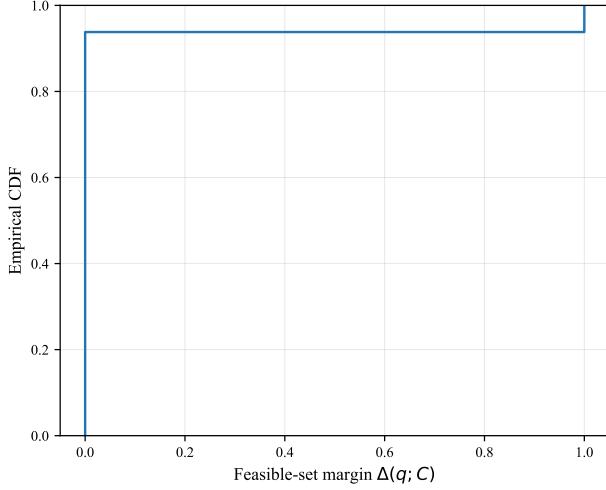


Figure A2. Margin distribution on MMR-BENCH. The feasible-set margin is highly concentrated near zero, indicating pervasive near-ties among candidate models.

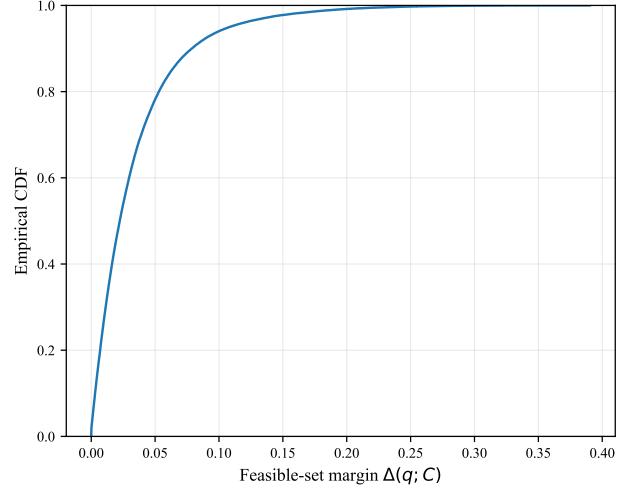


Figure A3. Margin distribution on MIXINSTRUCT. A large fraction of queries have small margins (e.g., $\approx 80\%$ within 0.05), suggesting that small-margin decision sensitivity is prevalent beyond RouterBench.

Results. Figure A1 compares three representative routers (MLPRouter, GraphRouter, and AvengersPro) against the Oracle under training-set evaluation. Even in this in-sample setting, routing collapse still occurs: learned routers increasingly concentrate their selections on the strongest model as the budget grows, while largely ignoring the remaining models. This indicates that routing collapse does not require train–test distribution shift and is therefore unlikely to be explained solely by out-of-sample generalization errors.

B.2. Experiment 2: Margin Distribution Beyond RouterBench

To verify that the small-margin regime is not an artifact of RouterBench, we repeat the margin analysis on two additional benchmarks: MMR-BENCH and MIXINSTRUCT. Across both datasets, we observe that near-ties between top candidates are common, indicating that decision sensitivity is a general property of routing benchmarks rather than a dataset-specific issue. In particular, on MMR-BENCH, the vast majority of queries exhibit exact ties among candidate models, while on MIXINSTRUCT, approximately 80% of queries have top-two margins within 0.05, suggesting that many queries admit multiple feasible models with comparable quality.

B.3. Why Small Margins Lead to Collapse

This section provides an intuitive explanation of why routing collapse emerges when small margins are prevalent. Under a budget constraint, routing always selects the top-ranked model among the feasible candidates. As a result, the routing outcome is determined by *relative comparisons* rather than absolute score values, and it can change whenever the

top-1 ordering changes.

A simple example. Consider a query q under some budget C where two feasible models j_1 and j_2 are the top contenders, and their ground-truth performances are nearly tied: $a_{j_1}(q) = 0.801$ and $a_{j_2}(q) = 0.800$, so the margin is $\Delta(q; C) = 0.001$. A learned router does not observe $a_j(q)$ at decision time and must rely on predicted scores. If the router’s predictions are perturbed by small, zero-mean errors (from finite data, stochastic optimization, model misspecification, etc.), then for some queries we may have $\hat{a}_{j_1}(q) = a_{j_1}(q) - 0.001$ and $\hat{a}_{j_2}(q) = a_{j_2}(q)$, which flips the ordering and makes the router select j_2 instead of j_1 . The key point is that, because the two candidates are extremely close, *a tiny perturbation is sufficient to change the discrete decision*.

From single flips to systematic collapse. When small margins dominate a benchmark, such top-1 flips become frequent across queries. This effect is amplified as budgets expand. As C increases, the feasible set typically contains stronger models more often, so the highest-ranked candidate is increasingly drawn from the top of the model pool. In a small-margin regime, many queries admit multiple near-optimal candidates within the feasible set; any small perturbation can tip the decision toward the strongest candidate. Aggregated over the query distribution, these frequent flips shift selection mass toward the strongest model, while smaller models are selected less and less. This produces the characteristic signature of routing collapse: the strongest model’s call rate rises sharply with budget, and the remaining models are progressively ignored.

Connection to the oracle-noise intervention. The oracle-noise experiment provides a controlled demonstration of this mechanism. Starting from oracle labels and injecting unbiased noise creates perturbations that only affect the *relative ordering* among close contenders, and we observe routing quality degrading rapidly as the noise increases. This behavior is consistent with the explanation above: when margins are small, comparison-based decisions are inherently sensitive, and small perturbations can accumulate into collapse.

B.4. Why does collapse concentrate on the strongest model (instead of an intermediate one)?

We provide a theory-level explanation under a standard *random-utility* view of routing decisions. Fix a query (we omit q for readability) and a budget C , and let the feasible set be $F(C) = \{j : c_j \leq C\}$. Assume the router ranks models by a (possibly learned) score subject to additive uncertainty,

$$\hat{a}_j = a_j + \varepsilon_j, \quad j \in F(C), \quad (9)$$

where a_j denotes the true (oracle) performance and $\{\varepsilon_j\}_{j \in F(C)}$ are i.i.d., continuous, and symmetric around 0. The router selects

$$\hat{j}(C) = \arg \max_{j \in F(C)} \hat{a}_j. \quad (10)$$

Let $j^*(C) \in \arg \max_{j \in F(C)} a_j$ be a (tie-broken) strongest feasible model in terms of true performance.

Key claim (maximum-mean winner). Under (9)–(10), the strongest feasible model has the largest selection probability:

$$\Pr[\hat{j}(C) = j^*(C)] \geq \Pr[\hat{j}(C) = j], \quad \forall j \in F(C). \quad (11)$$

Moreover, if $a_{j^*(C)} > a_j$ and the noise has a continuous density, the inequality is strict.

Proof sketch. For any candidate $j \in F(C)$, define the event that j is selected:

$$\begin{aligned} E_j &:= \left\{ \hat{a}_j \geq \hat{a}_k, \forall k \in F(C) \right\} \\ &= \left\{ \varepsilon_k - \varepsilon_j \leq a_j - a_k, \forall k \in F(C) \right\}. \end{aligned}$$

For each fixed j , the random vector $(\varepsilon_k - \varepsilon_j)_{k \in F(C)}$ has a distribution that does not depend on the values $\{a_\ell\}$ (it is fully determined by the i.i.d. noise law). Hence, $\Pr(E_j)$ is an *isotone* function of the thresholds $\{a_j - a_k\}_{k \in F(C)}$: increasing any threshold weakly increases the probability of the intersection in (B.4). If $j^*(C)$ maximizes a_j over $F(C)$, then for every $k \in F(C)$,

$$a_{j^*(C)} - a_k \geq a_j - a_k, \quad (12)$$

so the entire threshold vector for $E_{j^*(C)}$ dominates that of E_j coordinate-wise, implying (11). Strictness follows when at least one inequality is strict and the noise is continuous (the intersection boundary has measure zero).

Implication: instability does not “diffuse”—it concentrates on the top model. Equation (11) shows that with i.i.d. unbiased uncertainty and an argmax decision rule, the highest-true-performance feasible model is *always the single most likely winner* of the noisy competition. Therefore, errors in ranking do not spread probability mass uniformly across intermediate models; instead, whenever the strongest model is feasible, it attracts the largest share of selections. This yields an inherent bias toward the strongest model even when the perturbations are symmetric.

Why the effect strengthens as the budget increases. As C grows, the feasible set $F(C)$ expands monotonically and eventually contains the globally strongest model in the pool for an increasing fraction of queries. On those queries, (11) guarantees the strongest feasible model is the most probable choice. Consequently, the overall call frequency concentrates on the strongest model as C increases, producing the observed routing collapse.

Remark (ties and discrete metrics). When the evaluation metric induces many ties (or near-ties), the gaps $a_{j^*(C)} - a_j$ can be very small, which increases the sensitivity of (10) to the noise ε . However, (11) still holds: even in the small-margin regime, the strongest feasible model remains the most likely argmax, so instability manifests primarily as concentration on the top model rather than dispersion across mid-tier models.

C. Cost Prediction

We do not describe cost prediction in the main paper because it is not the focus of this work. Nevertheless, learned routers require an estimate of the per-query inference cost $c_j(q)$ at test time to enforce a user budget C . In this appendix, we summarize common practice and detail how we implement cost prediction in our experiments.

C.1. Cost Predictor Used in This Work

For all methods, we train a lightweight cost predictor to estimate $c_j(q)$ for each candidate model $j \in \mathcal{M}$ given the query representation. Specifically, we fit a two-layer MLP $g_\varphi^{(c)}$ that takes the query embedding \mathbf{q} as input and outputs a vector of predicted costs $\hat{\mathbf{c}}(q) = [\hat{c}_1(q), \dots, \hat{c}_K(q)]^\top$:

$$\hat{\mathbf{c}}(q) = g_\varphi^{(c)}(\mathbf{q}).$$

We train $g_\varphi^{(c)}$ with a standard squared-error objective

$$\mathcal{L}_{\text{cost}}(q) = \frac{1}{K} \sum_{j=1}^K (\hat{c}_j(q) - c_j(q))^2,$$

on the same training split used for router training. At inference time, routers enforce the budget by filtering feasible models using $\hat{c}_j(q) \leq C$, and then apply their own decision rule (e.g., selecting the highest-ranked model among feasible candidates).

Implementation details. The cost MLP uses a hidden size H with ReLU activation and is trained with the same optimizer and schedule as the corresponding router. We standardize costs using the training-set mean and variance and de-standardize predictions at inference time. In our benchmarks, this predictor is robust across datasets and its errors do not materially affect the resulting performance–cost curves.

C.2. Why This Treatment Is Reasonable

We support our treatment of cost prediction with two targeted checks. First, we verify that per-query cost is highly predictable from the query embedding. Figure A4 reports the test-set error of the two-layer MLP cost regressor, achieving an MSE on the order of 10^{-9} , indicating that cost estimation is effectively exact under our evaluation protocol. Second, we test whether cost estimation could be responsible for routing collapse by replacing predicted costs with *oracle* costs while keeping each router’s performance predictions (or ranking scores) unchanged. Figure A5 shows that this hybrid evaluation yields nearly identical routing behavior and collapse persists, indicating that improving cost estimates alone does not resolve the issue. Together, these results justify treating cost prediction as a non-bottleneck and focusing on performance learning and the objective–decision mismatch.

D. Routing Collapse Index

Curve-level metrics such as nAUC, P_s , and QNC summarize the global performance–cost frontier, but they cannot reveal whether a router makes *dominated* choices on individual queries. In particular, a router may appear strong simply by frequently selecting the most capable (and expensive) model, even when there exist cheaper models that perform at least as well on the same query. To explicitly quantify this failure mode, we define the *Routing Collapse Index (RCI)*, a per-query metric that measures how often and how severely a router over-selects expensive models.

D.1. Per-query dominated decisions

Let $\mathcal{M} = \{1, \dots, K\}$ be the pool of K candidate models, and let $\{q_n\}_{n=1}^N$ be the evaluation queries. For each query q_n and model $j \in \mathcal{M}$, define

$$a_{n,j} := a_j(q_n), \quad c_{n,j} := c_j(q_n),$$

as the observed performance and per-query monetary cost, respectively. Fix a routing policy π (e.g., $\pi = \pi_C$ under some budget C), and denote the selected model for query q_n by

$$m_n := \pi(q_n) \in \mathcal{M}.$$

We first define the set of models that are *strictly cheaper* than the selected one:

$$S_n = \{j \in \mathcal{M} : c_{n,j} < c_{n,m_n}\}, \quad X_n = |S_n|.$$

We also define the best achievable performance for query q_n over the full model pool:

$$a_n^* = \max_{j \in \mathcal{M}} a_{n,j}.$$

A router exhibits collapse on a query when it either (i) selects a model that is not performance-optimal, or (ii) selects a performance-optimal model but misses strictly cheaper alternatives that achieve the same performance.

D.2. Routing Collapse Index

We assign a per-query collapse score $s_n \in [0, 1]$ as follows:

$$s_n = \begin{cases} 1, & a_{n,m_n} < a_n^*, \\ \frac{K_n}{X_n}, & a_{n,m_n} = a_n^* \text{ and } X_n > 0, \\ 0, & a_{n,m_n} = a_n^* \text{ and } X_n = 0, \end{cases}$$

where

$$K_n = \left| \{j \in S_n : a_{n,j} \geq a_{n,m_n}\} \right|$$

counts how many strictly cheaper models match or exceed the selected model’s performance on the same query. Intuitively, $s_n = 1$ flags queries where the router fails to select any performance-optimal model. When the router is performance-optimal, s_n measures the fraction of strictly cheaper alternatives it unnecessarily ignores.

Finally, the Routing Collapse Index is the mean per-query collapse score:

$$\text{RCI} = \frac{1}{N} \sum_{n=1}^N s_n \in [0, 1],$$

where smaller values indicate less collapse and better utilization of cheaper models.

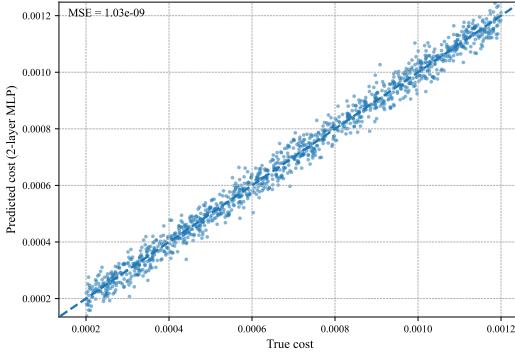


Figure A4. Cost prediction is near-exact. Test-set MSE of the two-layer MLP cost regressor is $\sim 10^{-9}$.

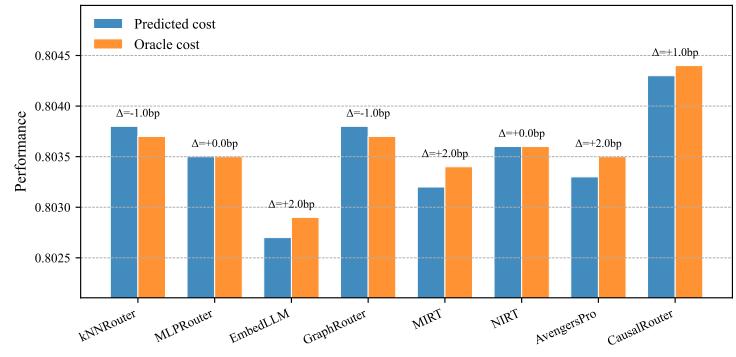


Figure A5. Oracle cost does not remove collapse. Using oracle costs with each router's predicted performance (or ranking scores) leaves routing behavior largely unchanged.

Practical notes. In our benchmarks, $a_{n,j}$ follows the task-specific evaluation protocol (e.g., exact match or model-graded correctness), and $c_{n,j}$ is computed from token pricing (or predicted costs when needed). For discrete metrics, ties are handled by the \geq comparison in K_n . If $X_n = 0$, then no strictly cheaper model exists and we set $s_n = 0$ by definition.

Illustrative examples. Example 1 (missed cheaper equivalent). Consider three models with costs $c_A < c_B < c_C$. For a query q , suppose $a_A(q) = a_B(q) = a_C(q) = 1$. If the router selects the most expensive model $m = C$, then $a^*(q) = 1$ and $S = \{A, B\}$ with $X = 2$. Since both cheaper models match the selected performance, $K = 2$ and $s = K/X = 1$.

Example 2 (dominated by a cheaper, better model). Using the same costs, suppose $a_A(q) = 0$, $a_B(q) = 1$, and $a_C(q) = 0$. If the router selects $m = C$, then $a^*(q) = 1$ and $a_m(q) = 0 < a^*(q)$, so $s = 1$.

E. Supplementary Experimental Results

EquiRouter Mitigate Routing Collapse. In the main paper, we focus on overall performance–cost trade-offs and do not break down how often each candidate model is selected. Here we provide an additional visualization of per-model call rates to illustrate how EquiRouter mitigates routing collapse. As shown in Fig. A6, EquiRouter substantially reduces the call rate of the strongest model (by roughly 50%) while increasing the utilization of smaller models, indicating more balanced routing decisions under the same budget sweep.

Out-of-Domain Robustness. In the main paper, we report results under the *in-domain* split, while real-world routing often encounters *out-of-domain* (OOD) inputs. We therefore additionally evaluate all methods on the OOD splits of two benchmarks, RouterBench and MMR-Bench, to assess

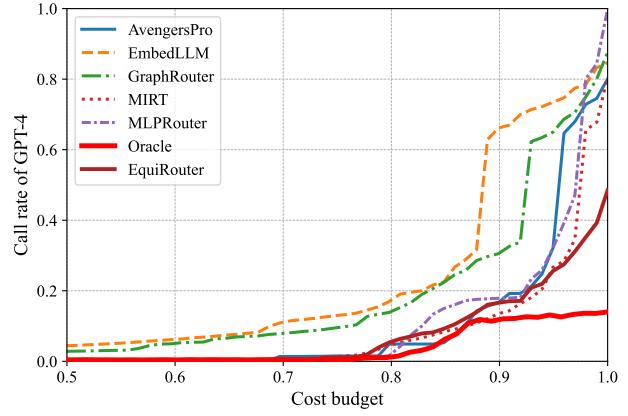


Figure A6. Per-model call rates under budget sweep. EquiRouter reduces the call rate of the strongest model by roughly 50% and increases the usage of smaller models, mitigating routing collapse.

robustness under distribution shift. As shown in Table A1, EquiRouter remains consistently strong across both benchmarks: it achieves the best overall performance–cost trade-off (highest nAUC with the lowest QNC among methods that report it) and simultaneously yields the lowest RCI. Moreover, EquiRouter also improves the strongest-model target success rate P_s on both benchmarks, indicating that its balanced utilization of smaller models does not come at the expense of final performance. Overall, these results suggest that EquiRouter generalizes beyond the in-domain regime and maintains stable, cost-aware routing decisions under OOD inputs.

F. Complexity Analysis

Let K be the number of candidate models, d_q the query embedding dimension, and D the trunk hidden dimension. EquiRouter computes a shared query trunk $z = f(\mathbf{q}) \in \mathbb{R}^D$ once per query. For each model j , it applies two linear projections $\phi(m_j) \in \mathbb{R}^{2D}$ and $\psi(m_j) \in \mathbb{R}^D$, forms a joint

Table A1. Out-of-domain evaluation on RouterBench and MMR-Bench. EquiRouter maintains strong performance–cost trade-offs and reduces RCI under distribution shift.

Router	Venue	RouterBench				MMR-Bench			
		nAUC (\uparrow)	QNC (\downarrow)	P_s (\uparrow)	RCI (\downarrow)	nAUC (\uparrow)	QNC (\downarrow)	P_s (\uparrow)	RCI (\downarrow)
kNNRouter	EMNLP '24	0.7502	1.0000	0.8116	0.7270	0.6694	/	0.7284	0.7684
MLPRouter		0.7532	1.0000	0.8116	0.7270	0.6394	/	0.6470	0.8138
EmbedLLM	[ICLR '24]	0.7467	/	0.8023	0.7305	0.6324	/	0.6850	0.7945
GraphRouter	[ICLR '25]	0.6771	1.0000	0.8116	0.7270	0.5864	/	0.7032	0.7960
MIRT		0.7491	0.9888	0.8119	0.7254	0.6635	/	0.7320	0.7894
NIRT	ACL '25	0.7491	1.0000	0.8116	0.7270	0.6232	/	0.7023	0.7754
AvengersPro	[DAI '25]	0.7524	/	0.7967	0.7349	0.6410	/	0.6736	0.7640
CausalRouter	[NeurIPS '25]	0.7524	/	0.8105	0.7694	0.6464	/	0.7305	0.8116
<i>EquiRouter</i>		0.7560	0.9568	0.8117	0.7064	0.6701	0.9847	0.7363	0.7249

feature $h_j \in \mathbb{R}^{4D}$, and evaluates a shared scoring head $g_\theta(h_j)$. The per-query time complexity is therefore

$$O(d_q D) + O(KD) + O(K \cdot \text{cost}(g_\theta)),$$

which is linear in K and dominated by the shared trunk and the per-model scoring. The additional memory beyond storing the fixed query encoder is $O(Kd_m)$ for the model embeddings and $O(D)$ for intermediate activations. In practice, K is modest and D is small, so EquiRouter adds negligible overhead relative to invoking an LLM.