CUT MY POOL INTO PIECES: RETHINKING MODEL POOL CONSTRUCTION FOR LLM ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

The enormous popularity of large language models (LLMs) has led to a vast and highly diverse landscape of models, varying in their strengths, weaknesses, and inference costs. As different models are suitable for different queries, LLM routing is increasingly applied to select an appropriate model for each query while balancing response quality and cost. Nonetheless, a router can only select among the models available in its pool, so the composition of that pool directly affects the routing system's quality-cost trade-off. Despite this, prior work has primarily focused on the router itself, paying considerably less attention to how the model pool should be constructed. In this work, we study model pool composition through pruning, systematically evaluating 10 pruning strategies across 27 benchmark tasks, 39 distinct models, and six router families. We find that well-pruned pools can preserve nearly all of the full pool's routing performance, while pruning strategies vary widely in their ability to serve low-cost budgets. Pool composition is particularly important for small pools, where its impact on routing performance exceeds that of router choice itself. Finally, pruning transfers well to unseen tasks, with performance close to that of in-task pruning.

1 INTRODUCTION

Less than a decade after the Transformer architecture was introduced (Vaswani et al., 2017), transformer-based large language models (LLMs) achieve remarkable performance across a wide range of tasks (Devlin et al., 2019; Team et al., 2024). Nevertheless, despite the broad landscape of available models (Aggarwal et al., 2024), there is no universally superior LLM (Chen et al., 2023). Instead, LLMs exhibit complementary strengths (Jiang et al., 2023), with different models performing best across different tasks and queries while also varying substantially in inference cost (Chen et al., 2023). This heterogeneity has motivated LLM routing, where a router dynamically selects an appropriate LLM from a model pool for each incoming query with the aim of balancing response quality and inference cost (Ong et al., 2025).

Recent work has proposed a variety of LLM routing approaches. For instance, Shnitzer et al. (2023) introduce a kNN-based router that estimates model performance for a new query based on semantically similar training queries (Hu et al., 2024). HybridLLM (Ding et al., 2024) and RouteLLM (Ong et al., 2025) learn binary routers that choose between a cheaper weak model and a more expensive strong model. Beyond pairwise routing, FrugalGPT (Chen et al., 2023) cascades a given query through multiple LLMs until the estimated response quality exceeds a threshold, whereas GraphRouter (Feng et al., 2025) represents tasks, queries, and LLMs as nodes in a heterogeneous graph to capture their interactions for LLM routing.

While these advances address the quality-cost trade-off (Song et al., 2025) and can lead to substantial cost savings (Ong et al., 2025), more complex routers do not consistently lead to better outcomes (Li, 2025). Recent work shows that simple kNN-based routers can be competitive with or even outperform substantially more complex routers while being more efficient (Li, 2025). Additionally, Srivatsa et al. (2024) report that the theoretical upper bounds of LLM routing exceed the performance of individual models, but implemented routers fail to realize this potential and remain similar to or slightly below the strongest individual LLMs. Lai & Ye (2026) further identify a systematic failure mode of existing routers, where learned routers tend to select the strongest and most expensive model even when cheaper models would suffice.

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.

Taken together, these findings motivate looking beyond the router itself. Since any router is constrained by the models available to it, model pool composition becomes a complementary design problem. This question is practically relevant because deployments may impose hard limits on the number of simultaneously active models (Li & Li, 2026), larger model pools can increase the data needed to train routers (Wang et al., 2026), and adding models to an existing pool may require additional labeling and retraining of routers (Jitkrittum et al., 2026).

While prior work has focused primarily on routing over a fixed model pool, recent studies have also begun to consider model pool construction. Li et al. (2026) observe diminishing returns with a growing model pool and find that routing over carefully selected subsets can outperform routing over larger random pools. Yuan et al. (2025) greedily prune the model pool by 27% based on similarities in model correctness while maintaining routing performance comparable to the full pool. Moreover, Shah & Shridhar (2025) propose a two-stage routing approach that first identifies a task-appropriate model pool and then cascades queries within it.

Together, these studies indicate that model pool composition affects routing outcomes and that smaller pools can retain much of the full pool's performance. However, existing approaches rely on criteria such as standalone accuracy or model similarity, without directly considering how much the remaining pool loses when a model is removed. This makes it difficult to distinguish models that are individually strong from those that are genuinely difficult for the pool to replace. Moreover, preserving performance alone is not sufficient when pruning a model pool. Removing cheap models may leave overall performance largely unchanged while reducing the pool's ability to operate at lower cost. A pool pruning strategy should therefore preserve both the full pool's quality-cost frontier and its range of supported budgets.

To address these gaps, we make the following contributions:

- We study model pool pruning across 27 benchmark tasks including 39 distinct models, with individual pools ranging from 12 to 37 models, six router families, and 10 pruning strategies. We thereby evaluate each pruned pool against the full pool's quality-cost frontier.
- We show that pruning can preserve routing performance, yet different strategies retain very different parts of the quality-cost frontier. Once inference cost matters, the models hardest to replace shift from the most accurate to the cheapest.
- We find that pool selection affects realized routing performance more than router choice, with the difference reaching 11–18$\times$ for the smallest pools. Well-pruned pools nevertheless preserve near full-pool routing performance with only a few models.
- We demonstrate that pool selection transfers well to unseen tasks, with transfer pruning achieving performance close to in-task pruning. The gap is largest in the most restrictive quality-only setting and narrows as pool size or cost awareness increases.

In the remainder of this paper, we review related work (Section 2), formalize the problem (Section 3), and describe our experimental setup (Section 4). We then present our results (Section 5) and summarize our main findings in Section 6.

2 Related Work

In this section, we review work on pool construction from its roots in algorithm selection, through ensemble selection and related pruning approaches for LLMs, to LLM routing, where the model pool itself has only recently received explicit attention.

Algorithm selection and portfolio construction: A direct abstraction of routed LLM systems was already formulated five decades ago by Rice (1976) as the algorithm selection problem. Here, a selector aims to identify the best-performing algorithm from a set of algorithms for a given problem instance (Kerschke et al., 2019). In addition to per-instance selection, prior work studies how the algorithm portfolio itself should be constructed. SATzilla (Xu et al., 2008) automatically constructs a portfolio for per-instance selection, PS-AAS (Kostovska et al., 2023) selects a diverse and non-redundant portfolio from a larger set of algorithms, and Hydra (Xu et al., 2010) iteratively expands a portfolio with algorithms that complement its current members. More recent work studies portfolio construction and algorithm selection jointly, including how portfolio size affects generalization

(Balcan et al., 2021). Building on this abstraction, we study which models should remain in a routed LLM portfolio and how their removal changes the attainable quality-cost frontier after rerouting.

Ensemble selection: Whereas algorithm selection chooses one algorithm for an instance, ensembles choose a subset whose outputs jointly determine the prediction. Prior work constructs compact ensembles by greedily adding the model that most improves the ensemble's validation metric (Caruana et al., 2004), balancing model quality and subset diversity (Sha et al., 2014), or using a Pareto formulation to maximize generalization performance while minimizing ensemble size (Qian et al., 2015). Additionally, dynamic ensemble selection forms an ensemble adaptively for each test instance (Ko et al., 2008; Oliveira et al., 2017). This distinction carries over to LLMs: LLM-TOPLA (Tekin et al., 2024) performs query-independent, diversity-based pruning of an LLM pool, whereas SelectLLM (Maurya et al., 2025), SkyLLM (Zhao & Zhu, 2025), and ThriftLLM (Huang et al., 2025) select a subset based on the input query. Despite these advances, ensemble selection targets a different deployment scenario. Specifically, it aggregates multiple model outputs to produce a final prediction, whereas in our setting the router executes only one model per query.

LLM routing and pool composition: Unlike LLM ensembles, LLM routing systems select and execute a single model per query, often balancing response quality and inference cost. Prior work comprises binary routers such as RouteLLM (Ong et al., 2025) and HybridLLM (Ding et al., 2024), kNN-based routing as proposed by Shnitzer et al. (2023), and cascading approaches such as FrugalGPT (Chen et al., 2023). UniRoute (Jitkrittum et al., 2026) further supports dynamic model pools and can route to new LLMs unseen during router training. Beyond routing, recent work has also begun to investigate the model pool itself. Li et al. (2026) compare random model pools with pools comprising the most accurate models. They find that the latter can outperform random pools despite containing fewer models and observe diminishing returns as pool size increases. LOCUS (Patel et al., 2026) learns embeddings that represent model capabilities and uses them in a cluster-based approach to construct a smaller pool. They find that a pool of 15 models achieves routing accuracy comparable to the full pool of 112 models. Yuan et al. (2025) and Schönwälder et al. (2026) use model correctness to identify and remove redundant LLMs. The former greedily reduce a model pool by 27% based on model accuracy and correctness overlap and roughly match full-pool routing performance. The latter propose two pruning heuristics for eliminating LLM redundancy. Directly combining routing with pool composition, Shah & Shridhar (2025) propose a two-stage system in which a model pool is selected based on the query and subsequently used for routing. StageRoute (Li & Li, 2026) instead models a typical deployment scenario by periodically updating the deployed model pool as new models become available and routing queries within the current pool.

Building on a long history of pool construction, prior work on LLM routing has mainly shown that model pools contain considerable redundancy. The open question is therefore no longer simply whether a pool can be reduced, but what is actually lost in doing so. Which capabilities are lost when models are pruned, which models are replaceable, how do these losses affect realized routing performance, and does effective pruning generalize to unseen tasks? We study these questions systematically across a broad range of tasks, models, pruning strategies, and router families.

3 Problem Formulation and Evaluation

In this section, we formalize model-pool pruning for LLM routing and define our evaluation framework. We evaluate pruned pools both at fixed cost preferences and across the quality-cost frontier. We then measure how much of this attainable performance learned routers can realize.

3.1 Problem Setup

Let $\mathcal{P} = \{m_1, \ldots, m_M\}$ be a pool of M candidate LLMs. For each query x drawn from a task-specific distribution $\mathcal{D}$ over $\mathcal{X}$, executing model m yields two measurable quantities, a quality score $q(x, m) \in [0, 1]$ and an inference cost $c(x, m) \geq 0$. To model the trade-off between quality and cost, we define the utility of selecting model m for query x at a cost preference $\lambda \geq 0$ as $u_\lambda(x, m) = q(x, m) - \lambda c(x, m)$. Before serving queries, pruning selects a subset $\mathcal{S} \subseteq \mathcal{P}$. A router $\pi_\mathcal{S} : \mathcal{X} \to \mathcal{S}$ then maps each query to one of the retained models. Consequently, pruning determines which models are available, while routing determines which retained model answers each query.

3.2 Evaluating Pruned Pools

We evaluate pruned pools from two perspectives, one by measuring their performance at fixed cost preferences and the other by assessing how well they preserve the quality-cost frontier.

Fixed-Preference Evaluation: For a subset $\mathcal{S} \subseteq \mathcal{P}$, we define its oracle value $V_\lambda(\mathcal{S})$ at cost preference λ as the expected utility obtained by routing each query to the highest-utility model in $\mathcal{S}$:

$$V_\lambda(\mathcal{S}) = \mathbb{E}_{x\sim\mathcal{D}}\left[\max_{m\in\mathcal{S}} u_\lambda(x,m)\right].$$

To make this value comparable across tasks, we normalize it using $\mathrm{retention}_\lambda(\mathcal{S})$ between the task-specific full-pool oracle and a random-routing anchor $R_\lambda(\mathcal{P})$, representing the utility expected when queries are assigned uniformly at random to models in the full pool:

$$\mathrm{retention}_\lambda(\mathcal{S}) = \frac{V_\lambda(\mathcal{S}) - R_\lambda(\mathcal{P})}{V_\lambda(\mathcal{P}) - R_\lambda(\mathcal{P})}, \qquad \text{where} \qquad R_\lambda(\mathcal{P}) = \frac{1}{|\mathcal{P}|}\sum_{m\in\mathcal{P}} \mathbb{E}_{x\sim\mathcal{D}}\left[u_\lambda(x,m)\right].$$

A retention of one recovers the full-pool oracle value, while zero matches random routing.

Full-Frontier Evaluation: Retention evaluates a subset at a specific cost preference λ. To assess whether pruning preserves performance across different quality-cost trade-offs, we additionally compare the quality-cost frontiers of the full and pruned pools. For a pool $\mathcal{S}$, the quality-cost frontier is the upper concave envelope of the mean cost-quality points attainable under oracle routing.

To quantify frontier loss after pruning, we define the *budget gap* $\mathrm{BG}(\mathcal{S})$ as the average vertical gap between the full-pool and sub-pool quality-cost frontiers. Let $Q_\mathcal{S}(b)$ denote the maximum quality attainable by routing over $\mathcal{S}$ at an average cost no greater than b:

$$\mathrm{BG}(\mathcal{S}) = \frac{1}{b_{\mathrm{hi}} - b_{\mathrm{lo}}}\int_{b_{\mathrm{lo}}}^{b_{\mathrm{hi}}} \left(Q_\mathcal{P}(b) - Q_\mathcal{S}(b)\right)\, db.$$

Here, b_{lo} and b_{hi} are the endpoints of the full-pool training frontier and define a common budget range across subsets. If $\mathcal{S}$ cannot operate at budget b, we set $Q_\mathcal{S}(b) = 0$. Thus, $\mathrm{BG}(\mathcal{S})$ measures at each budget how high the sub-pool frontier reaches relative to the full-pool frontier. It is large when pruning lowers attainable quality across many budgets or makes low-cost budgets infeasible.

3.3 Realized Router Performance

The preceding metrics characterize the performance attainable by a pruned pool, while a learned router $\pi_\mathcal{S}$ may realize only part of it. We therefore define its achieved utility $A_\lambda(\pi_\mathcal{S})$ and its normalized end-to-end performance $end2end_\lambda(\pi_\mathcal{S})$ as

$$A_\lambda(\pi_\mathcal{S}) = \mathbb{E}_{x\sim\mathcal{D}}\left[u_\lambda(x,\pi_\mathcal{S}(x))\right], \quad end2end_\lambda(\pi_\mathcal{S}) = \frac{A_\lambda(\pi_\mathcal{S}) - R_\lambda(\mathcal{P})}{V_\lambda(\mathcal{P}) - R_\lambda(\mathcal{P})},$$

An $end2end$ value of one matches the full-pool oracle, while zero matches random routing. The remaining gap to the full-pool oracle can be decomposed into pruning loss pl, caused by removing models, and routing loss rl, caused by suboptimal routing within the retained pool:

$$pl_\lambda(\mathcal{S}) = 1 - \mathrm{retention}_\lambda(\mathcal{S}), \quad rl_\lambda(\pi_\mathcal{S}) = \frac{V_\lambda(\mathcal{S}) - A_\lambda(\pi_\mathcal{S})}{V_\lambda(\mathcal{P}) - R_\lambda(\mathcal{P})}.$$

By construction, $1 = pl_\lambda(\mathcal{S}) + rl_\lambda(\pi_\mathcal{S}) + end2end_\lambda(\pi_\mathcal{S})$ holds.

4 EXPERIMENTAL SETUP

This section summarizes our experimental setup, covering the data and cost normalization, the pruning strategies and router families, and the two experimental tracks.

4.1 DATA AND INFERENCE COSTS

We use LLMRouterBench as our data source as it covers a broad range of tasks and language models. After data cleaning like removing query-model pairs with missing quality, our experimental corpus comprises 27 tasks, 39 distinct LLMs, and approximately 480,000 query-model pairs, with the number of available models ranging from 12 to 37 across tasks. As LLMRouterBench only provides costs for a subset of the queries and models (e.g., all costs for Qwen2.5-Coder-7B-Instruct are zero), we compute costs for each query using input and output tokens together with token costs per model. For each task, we construct five randomly shuffled train/val/test splits using a 60/10/30 ratio. Details regarding models, and cost computation are provided in Appendix A.1.

Because inference costs and attainable quality ranges vary across tasks, a fixed λ-grid does not represent comparable cost preferences. For example, mean query cost ranges from 0.014USD for AIME to 0.00008USD for ARC-C, a roughly $175\times$ difference that yields very different quality–cost trade-offs for the same λ. We therefore normalize cost preference by each task's attainable ranges using $\lambda_t = t \cdot scale$, with $scale = \frac{\Delta q}{\Delta c}$, where Δq and Δc denote the full pool's task-specific quality and cost spans. We evaluate a broad grid of relative preferences, $t \in \{0, 0.5, 1, 2, 4, 8, 16, 32\}$, where $t = 0$ considers quality only, $t = 1$ weighs the task's full attainable cost span equally with its quality span, and $t > 1$ represents increasingly cost-sensitive settings.

4.2 PRUNING AND ROUTING

Our experiments involve ten pruning strategies comprising 18 different configurations. They cover simple ranking pruners (random, cheapest-first, highest average quality, and highest average utility), redundancy-based pruners that compute overlap between models' quality (overlap-quality) or between their wins and losses (overlap-conditional-win-loss), objective-aware pruners that greedily remove models to minimize utility loss (greedy-regret, greedy-regret-feasible) or budget gap (frontier-greedy), and a greedy approach (greedy-forward-utility) that adds models to the pool rather than removing them. Full definitions, and configurations are given in Appendix A.2.

We evaluate six router families spanning seven configurations. As simple baselines, we use a random router with independent seeds and a best-model router that selects the model with the highest mean training utility. Inspired by Shnitzer et al. (2023), KNN predicts the utility of each available model from similar training queries. The matrix-factorization router (MF) is a simple learner that uses query embeddings to predict either the quality or utility of each available model, depending on the configuration. MF is inspired by RouteLLM (Ong et al., 2025). However, whereas RouteLLM chooses between two models and is therefore not directly applicable to our setting, MF predicts a score for every model in the pool. We further include RouterDC (Chen et al., 2024) and GraphRouter (Feng et al., 2025) as computationally expensive neural routing approaches. All query-dependent routers use the same BGE-M3 query embeddings. Further details are given in Appendix A.3.

4.3 EXPERIMENTAL TRACKS

Our experiments consist of two consecutive tracks. In the first, *pool-selection* track, we evaluate all 18 pruning configurations for every task and each of its five shuffled train/val/test splits. Starting from the task-specific full pool of $k_{\max}$ models, each pruner iteratively removes models until the minimum pool size $k_{\min} = 3$ is reached. Cost-dependent pruners are fitted per λ (derived from t; see Subsection 4.1), λ-independent pruners (e.g., cheapest-first) are fitted once. The resulting pools are then evaluated with their query-wise oracle on the corresponding test queries using retention and budget gap. The second track, *router realization*, measures how much of the performance attainable with each pruned pool can be achieved by learned routers. Because router training is costly, we restrict the evaluation to pool sizes from a reduced ladder, $k \in \{3, 4, 5, 6, 8, 10, 13, 17, 22, 28, k_{max}\}$. We report $end2end$ utility together with pruning and routing losses.

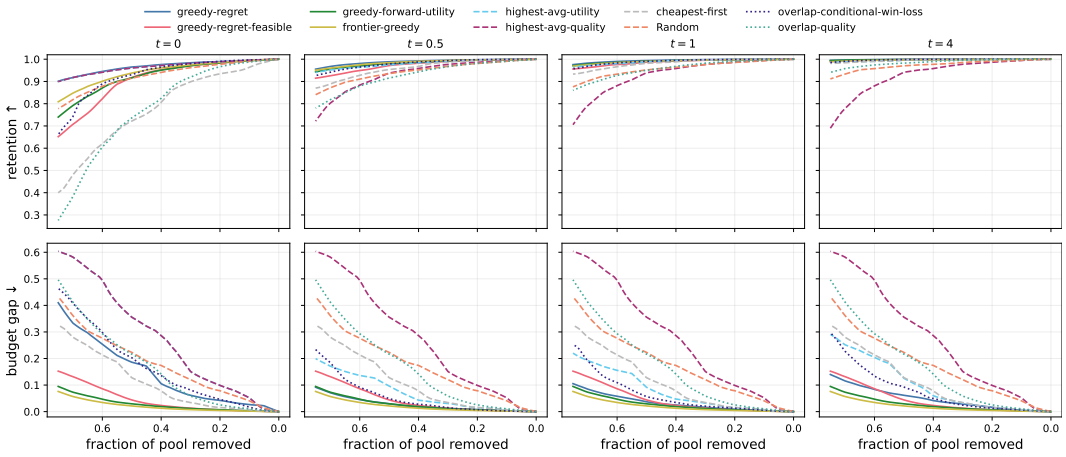

Figure 1: Retention (top) and budget gap (bottom) for selected pruning strategies and cost preferences (t) across fractions of models removed; further results are reported in Appendix A.5.

Across both tracks, we also study the transfer of pruning to unseen tasks. For each target task, a pool is selected using the combined splits of up to 26 other tasks, excluding any that share queries with the target, and evaluated on the target task. For router realization, we restrict the transferred pools to $k \in \{3, 5, 8\}$ to limit computational cost and use them to train and evaluate routers on the target task. Further details, including the construction of the reduced k-ladders, the exact pruning configurations, and the transfer setting, are provided in Appendix A.4.

5 Results

This section organizes the results around three questions: (1) what is lost when model pools are pruned (Section 5.1), (2) whether the resulting pools preserve routing performance (Section 5.2), and (3) whether effective pool selection transfers to previously unseen tasks (Section 5.3).

5.1 What Is Lost When Pruning Model Pools?

In the pool-selection track, we evaluated a total of 613,440 pools created by the 18 different pruner configurations applied to all 27 benchmark tasks, 5 shuffled splits, and 8 relative cost preferences, ranging from the task-specific full pool to a minimum pool size of $k_{\min} = 3$. Since pool sizes vary across tasks, Figure 1 reports results by the fraction of models removed, allowing a fair comparison across tasks. Each pruned pool is evaluated using both retention and budget gap.

First, we ask how much utility is retained after pruning relative to the full model pool. Measured by retention in the figure's upper row, the results directly reproduce the substantial pool redundancy reported in prior work. At cost preference $t = 0$, where utility is determined solely by quality, the employed pruning strategies preserve an average retention of 0.861 after removing 50% of each task's full pool. This effect becomes even stronger once cost is taken into account. For $t > 0$, removing 70% of the models still yields an average retention of 0.960 across all 18 tested pruning configurations. Even under the strongest compression tested ($k = 3$) at $t = 1$, highest-avg-utility and greedy-regret retain 0.948 and 0.957, respectively.

These results suggest that model pools can be pruned extensively with little loss in utility. Nonetheless, high retention does not indicate whether the pruned pool can still operate across the same range of costs or becomes infeasible under low-cost budgets. We therefore evaluate the same pools using budget gap, which measures how much of the full quality-cost frontier is lost after pruning. The lower row of Figure 1 provides this complementary view and reveals larger differences between pruning strategies. While several pruning strategies achieve similarly high retention, they differ considerably in how well they preserve the quality-cost frontier. At $t = 1$, for instance, highest-avg-utility, greedy-regret, and frontier-greedy all maintain high retention ranging between 0.964 and

Table 1: Infeasible share after removing a fixed fraction of each task-specific model pool, averaged across tasks and splits; additional results are reported in Appendix A.5.

	$t = 0$			$t = 1$		
Pruning strategy / models removed	50%	60%	70%	50%	60%	70%
highest-avg-quality	0.387	0.516	0.630	0.387	0.516	0.630
highest-avg-utility	0.387	0.516	0.630	0.038	0.084	0.097
greedy-regret	0.154	0.214	0.273	0.002	0.002	0.007
frontier-greedy	0.002	0.003	0.005	0.002	0.003	0.005

0.976 even when removing 75% of each pool, while frontier-greedy consistently yields the smallest budget gaps (0.076 vs 0.220 highest-avg-utility and 0.105 greedy-regret). Hence, pruning quality cannot be characterized by retained utility alone. Two pools may achieve similar retention but vary in which parts of the full quality-cost frontier they preserve.

This difference raises the question of what causes the observed frontier loss. To answer this, we measure the infeasible share, defined as the fraction of the full pool's budget range in which the subpool cannot operate because it contains no model at the corresponding cost level. Table 1 reports this measure for $t = 0$ and $t = 1$ across increasing pruning rates. At $t = 1$, highest-avg-utility achieves 0.974 retention after removing 70% of the models, yet roughly 10% of the original budget range is no longer feasible. Greedy-regret and frontier-greedy achieve similarly high retention of 0.985 and 0.977, respectively, while leaving only less than 1% of the budget range infeasible. This difference is particularly highlighted under the quality-only preference $t = 0$. Here, highest-avg-quality makes 39% of the full-pool budget range infeasible after removing 50% of the models, with this share increasing at higher removal rates. In contrast, frontier-greedy preserves nearly the entire feasible budget range across the same pruning rates. Thus, the budget-gap differences in Figure 1 reflect a specific deployment limitation of pruning. A pool may retain nearly all utility after pruning but no longer contain a model cheap enough to operate under certain inference-cost budgets. Consequently, utility redundancy does not imply budget redundancy.

5.1.1 Which Models Are Actually Replaceable?

The previous results show that pruning strategies can remove the same number of models while preserving very different parts of the quality-cost frontier, pointing to differences in how replaceable individual models are. To quantify this, we define the removal cost of model m as $RC_\lambda(m) = V_\lambda(P) - V_\lambda(P \setminus m)$, i.e., the loss in full-pool oracle utility after removing m. For each task and split, we rank models by cost and accuracy, group them into thirds, and report each group's share of total removal cost. Figure 2 summarizes these shares across tasks.

At $t = 0$, where utility depends only on quality, removal cost is concentrated among the most accurate models, with the most accurate third accounting for 57.5% of total removal cost. Once inference cost enters the objective, this pattern reverses. The cheapest third accounts for 55.5% of total removal cost already at $t = 0.5$, increasing to 61.7% at $t = 1$ and 67.8% at $t = 4$. At the same time, contributions from accurate and expensive models decline sharply. This shift explains why quality-oriented pruning can preserve high retention while still damaging the low-cost part of the frontier. Inexpensive models may be weaker individually, but together they can preserve both high utility and low-cost operating points. Model value is therefore not determined by individual model quality alone, but by the role a model plays within the pool.

Taken together, these results show that aggressive pruning is often possible without substantial loss in utility, but that high retention does not imply preservation of the full quality-cost frontier. Pruned pools with similar utility can differ in their budget gaps and in their range of feasible budgets. These differences arise because model value is objective-dependent. Highly accurate models are hardest to replace when quality alone matters, whereas inexpensive models become critical once inference cost is considered. Thus, the central challenge in pool pruning is not simply how many models can be removed, but which parts of the quality-cost frontier must remain available for deployment.

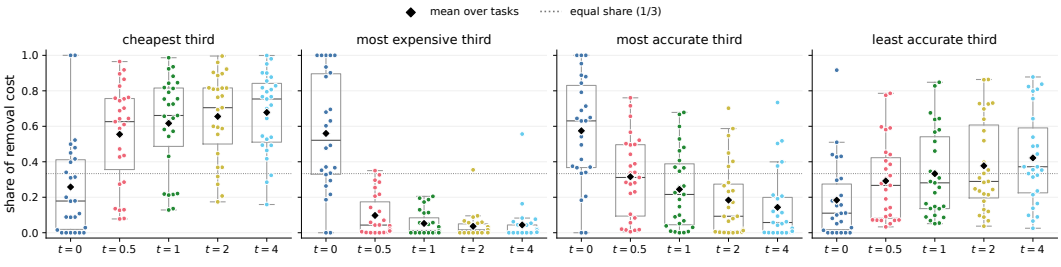

Figure 2: Share of total removal cost across model groups divided into thirds by cost or accuracy.

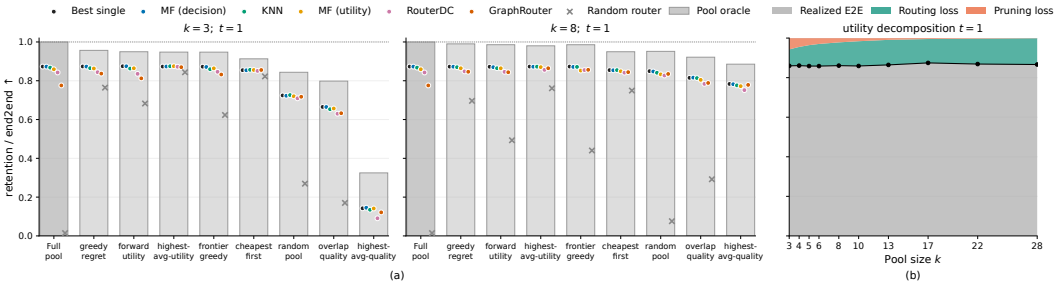

Figure 3: Router realization at $t = 1$. (a) Pool retention and realized $end2end$ utility for $k = 3$ and $k = 8$; (b) decomposition into $end2end$, routing loss, and pruning loss across pool sizes.

5.2 Does Pruning Preserve Realized Routing Performance?

While we previously measured the potential of pruned pools, we now use them in the router realization track to train and evaluate actual routers. Specifically, we fit six router families spanning seven configurations and apply them to all produced pools for a selection of k values ($k \in \{3, 4, 5, 6, 8, 10, 13, 17, 22, 28, k_{max}\}$). We then evaluate realized routing performance using $end2end$ utility and decompose the remaining gap into pruning and routing loss. Figure 3(a) compares each pool's oracle retention with the $end2end$ utility realized by the evaluated routers at $t = 1$. At $k = 3$, cost-aware pruning methods like greedy-regret or highest-avg-utility retain between 0.948 and 0.957 of the full-pool oracle advantage. At $k = 8$, their retention increases to between 0.980 and 0.990. In contrast, highest-avg-quality retains only 0.325 at $k = 3$ and remains the weakest method at $k = 8$, with 0.885 retention. Despite their lower oracle ceilings, pruned pools with high retention achieve $end2end$ utility comparable to the full pool. Across the six non-random routers, full-pool $end2end$ utility ranges from 0.776 to 0.873. For greedy-regret pruning, five of six routing configurations remain within approximately 0.006 of their full-pool performance (GraphRouter improves more substantially). Thus, effective pruning preserves average $end2end$ utility across the evaluated cost preferences (remaining results in Appendix A.6). However, once a pruning strategy such as highest-avg-quality has reduced a pool's retention, routing cannot recover the lost utility.

To isolate the effect of pool size once pruning already preserves high oracle utility, Figure 3(b) averages the cost-aware pruning methods (greedy-regret, forward-utility, highest-avg-utility, frontier-greedy, cheapest-first) and the non-random routers. At $k = 3$, mean $end2end$ utility is 0.860, with a pruning loss of 0.057 and a routing loss of 0.083. Increasing the pool to $k = 8$ changes $end2end$ utility by only 0.001, from 0.860 to 0.861. Thus, the additional models improve the oracle ceiling but provide little additional realized utility under the evaluated routing policies.

Increasing pool size mainly helps when pruning removes valuable models. Once an effective pruning strategy already retains high oracle utility, additional models increase the oracle ceiling but yield little $end2end$ utility gain because routers do not exploit the added headroom. Moreover, realized utility is more sensitive to pool selection than to non-random router choice, with 11–18$\times$ larger variation across pruning configurations at $k = 3$ across the eight cost preferences. This difference narrows as pool size increases.

Table 2: Relative shortfall (%) of transfer versus in-task selection using highest-avg-utility.

	t=0		t=1		t=4	
Pool size	retention	$end2end$	retention	$end2end$	retention	$end2end$
$k = 3$	12.5	13.1	1.9	2.0	1.3	1.6
$k = 5$	4.8	5.4	1.5	2.1	0.5	0.8
$k = 8$	2.3	1.3	0.8	0.7	0.4	0.0

5.3 Does Pool Pruning Transfer to Unseen Tasks?

While the previous subsections study pool pruning using data from the target task (in-task), we additionally investigate whether pool selection transfers to unseen tasks in a deployment-relevant setting (transfer). We treat each of the 27 benchmark tasks in turn as the target task, while the remaining 26 tasks (sometimes fewer, as we exclude source tasks with queries overlapping the target) serve as source tasks on which the different pruners are fitted. Subsequently, the resulting pruned pool is then used to train and evaluate routers on the target task.

Table 2 reports the shortfall of transfer pruning relative to in-task pruning for highest-avg-utility, which achieves the highest average $end2end$ utility across routers in both settings (other pruning strategies show similar patterns; further results are in Appendix A.7). We compare the retention of the pruned pools and the $end2end$ utility of the non-random routers, averaged across splits and tasks and, for $end2end$ utility, additionally across routers. At $t = 0$ and $k = 3$, the retention of the transfer-pools is 12.5% lower than that of pools fitted directly on the target task. However, this shortfall generally decreases with increasing cost awareness, reaching only 1.3% at $k = 3$ and $t = 4$. It also decreases with increasing pool size, reaching 2.3% at $k = 8$ and $t = 0$. The same trend is observed for $end2end$ utility and for the other evaluated pruning strategies in the transfer setting. Across these strategies and $t > 0$, the largest shortfall relative to the best in-task strategy is 5.9% in retention and 7.7% in $end2end$ utility (details and further results are provided in Appendix A.7).

In summary, except for the most restrictive and cost-insensitive setting ($k = 3$, $t = 0$), transfer pruning remains close to in-task pruning in both retention and $end2end$ utility. Moreover, the shortfall decreases as the pool size and cost awareness increase. Thus, target-task labels become less important for pool pruning as more models are retained or inference cost receives greater weight. Note that we slightly adjusted the pruning strategies for the transfer setting to account for different pool sizes across source tasks and previously unseen target-task models (see Appendix A.7 for details).

6 Conclusion

Our results show that the model pool is not merely an input to an LLM router, but a central component influencing the routing system's achievable performance. This becomes particularly clear for small pools, where pool selection affects realized routing performance far more than the choice among the evaluated non-random routers. Although effective pruning can preserve nearly all realized routing performance, which models remain in the pool matters. In particular, strategies with similar retained utility can differ in the quality–cost frontier and budget ranges they preserve. At the level of individual models, those hardest for the remaining pool to replace shift from the most accurate to the cheapest once inference cost matters. Beyond the in-task setting, effective pool pruning also generalizes well across tasks, suggesting that compact deployment pools need not require extensive target-task data. Taken together, these findings show that effective LLM routing depends not only on how queries are routed, but also on which models are made available to the router. Model pool construction should therefore be treated as a core deployment decision rather than a fixed prerequisite of routing.

AI use statement

Generative AI tools were used for grammar and language polishing, code review, and assistance with code development. All AI-assisted code was reviewed and validated by the authors.

Reproducibility statement

We provide the code used for all experiments in this paper in our anonymized public repository at `https://anonymous.4open.science/r/CutMyPoolIntoPieces-0180` to support reproducibility.

References

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. Automix: Automatically mixing language models. *Advances in Neural Information Processing Systems*, 37:131000–131034, 2024.

Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Generalization in portfolio-based algorithm selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 12225–12232, 2021.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 18, 2004.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. Routerdc: Query-based router by dual contrastive learning for assembling large language models. *Advances in Neural Information Processing Systems*, 37:66305–66328, 2024.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks Lakshmanan, and Ahmed H Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. In *International Conference on Learning Representations*, volume 2024, pp. 41348–41366, 2024.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. In *International Conference on Learning Representations*, volume 2025, pp. 26186–26203, 2025.

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. *arXiv preprint arXiv:2403.12031*, 2024.

Keke Huang, Yimin Shi, Dujian Ding, Yifei Li, Yang Fei, Laks Lakshmanan, and Xiaokui Xiao. Thriftllm: On cost-effective selection of large language models for classification queries. *arXiv preprint arXiv:2501.04901*, 2025.

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 (Volume 1: Long Papers)*, pp. 14165–14178, 2023.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, et al. Universal model routing for efficient llm inference. In *International Conference on Learning Representations*, volume 2026, pp. 10169–10218, 2026.

Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019. doi: 10.1162/evco_a_00242.

Albert HR Ko, Robert Sabourin, and Alceu Souza Britto Jr. From dynamic classifier selection to dynamic ensemble selection. *Pattern recognition*, 41(5):1718–1731, 2008.

Ana Kostovska, Gjorgjina Cenikj, Diederick Vermetten, Anja Jankovic, Ana Nikolikj, Urban Skvorc, Peter Korosec, Carola Doerr, and Tome Eftimov. Ps-aas: Portfolio selection for automated algorithm selection in black-box optimization. In *International Conference on Automated Machine Learning*, pp. 11–1. PMLR, 2023.

Guannan Lai and Han-Jia Ye. When routing collapses: On the degenerate convergence of llm routers. *arXiv preprint arXiv:2602.03478*, 2026.

Hao Li, Yiqun Zhang, Zhaoyan Guo, Chenxu Wang, Shengji Tang, Qiaosheng Zhang, Yang Chen, Biqing Qi, Peng Ye, Lei Bai, et al. Llmrouterbench: A massive benchmark and unified framework for llm routing. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 37733–37754, 2026.

Shaoang Li and Jian Li. Near-optimal online deployment and routing for streaming llms. In *International Conference on Learning Representations*, volume 2026, pp. 90170–90202, 2026.

Yang Li. Rethinking predictive modeling for llm routing: When simple knn beats complex learned routers. *arXiv preprint arXiv:2505.12601*, 2025.

Kaushal Kumar Maurya, KV Aditya Srivatsa, and Ekaterina Kochmar. Selectllm: Query-aware efficient selection algorithm for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20847–20863, 2025.

Dayvid VR Oliveira, George DC Cavalcanti, and Robert Sabourin. Online pruning of base classifiers for dynamic ensemble selection. *Pattern Recognition*, 72:44–58, 2017.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

Shivam Patel, William Cocke, and Gauri Joshi. Locus: Low-dimensional model embeddings for efficient model exploration, comparison, and selection. *arXiv preprint arXiv:2601.21082*, 2026.

Chao Qian, Yang Yu, and Zhi-Hua Zhou. Pareto ensemble pruning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

John R. Rice. The algorithm selection problem. volume 15 of *Advances in Computers*, pp. 65–118. Elsevier, 1976. doi: https://doi.org/10.1016/S0065-2458(08)60520-3. URL `https://www.sciencedirect.com/science/article/pii/S0065245808605203`.

Erik Schönwälder, Claudio Hartmann, and Wolfgang Lehner. Rejectors in the wild: Deployment barriers for llm rejectors. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 4127–4138, 2026.

Chaofeng Sha, Keqiang Wang, Xiaoling Wang, and Aoying Zhou. Ensemble pruning: A submodular function maximization perspective. In *International Conference on Database Systems for Advanced Applications*, pp. 1–15. Springer, 2014.

Soham Shah and Kumar Shridhar. Select-then-route: Taxonomy guided routing for llms. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 425–441, 2025.

Tal Shnitzer, Anthony Ou, Mírian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. Large language model routing with benchmark datasets. *arXiv preprint arXiv:2309.15789*, 2023.

Wei Song, Zhenya Huang, Cheng Cheng, Weibo Gao, Bihan Xu, GuanHao Zhao, Fei Wang, and Runze Wu. Irt-router: Effective and interpretable multi-llm routing via item response theory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15629–15644, 2025.

Kv Aditya Srivatsa, Kaushal Kumar Maurya, and Ekaterina Kochmar. Harnessing the power of multiple minds: Lessons learned from llm routing. In *Proceedings of the Fifth Workshop on Insights from Negative Results in NLP*, pp. 124–134, 2024.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Selim Furkan Tekin, Fatih Ilhan, Tiansheng Huang, Sihao Hu, and Ling Liu. Llm-topla: Efficient llm ensemble by maximising diversity. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 11951–11966, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Zijie Wang, Xinyu Yan, Che Wang, Zeng Zihao, Lei Xiao, and Wei Yang Bryan Lim. Semirouter: Sparse-data enhanced routing for adaptive multi-llm system. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4910–4921, 2026.

Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.

Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pp. 210–216, 2010.

Jiayi Yuan, Yifan Lu, Rixin Liu, Yu-Neng Chuang, Hongyi Liu, Shaochen Zhong, Yang Sui, Guanchu Wang, Jiarong Xing, and Xia Hu. Who routes the router: Rethinking the evaluation of llm routing systems. 2025.

Heng Zhao and Yifei Zhu. Skyllm: Cross-llm-apis federation for cost-effective query processing. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20864–20873, 2025.

A Appendix

A.1 Benchmark tasks and cost reconstruction

We use the released LLMRouterBench query-model pairs from the corresponding GitHub repository (`https://github.com/ynulihao/LLMRouterBench.git`) as our data source and exclude the routing pseudo-model `openrouter`. Within each task, we retain only models for which a quality score and a finite reconstructed cost are available for every query, yielding a dense query-by-model matrix. The resulting corpus comprises 27 tasks, 39 models, and 479,605 query-model pairs. Because some recorded costs are zero and some models have no common pricing basis, we reconstruct costs using model-specific token prices. For each query-model pair, we compute the inference cost in USD as

$$\text{cost} = \frac{\text{input tokens} \cdot \text{input price} + \text{output tokens} \cdot \text{output price}}{10^6},$$

where prices are given in USD per million tokens. We clip negative token counts to zero. For API models, we infer input and output token prices by matching token counts to the reported costs. When several datasets give an exact match, we use the most common price pair. For GLM-4.6, we instead use the prices reported in LLMRouterBench because the recovered rates differ across datasets. For all local 7-9B models, we use estimated token prices. Our 8B reference prices are 0.06USD per million input tokens and 0.09USD per million output tokens, scaled by each model's parameter count divided by 8B. These reference prices are the midpoints of the input and output price ranges of three comparable open-weight models on DeepInfra, accessed in June 2026: Llama-3.1-8B (0.02USD/0.05USD), Llama-3.1-8B-Turbo (0.02USD/0.03USD), and Qwen3.5-9B

Table 3: Model prices in USD per million tokens. Rates are recovered from recorded costs, adopted from LLMRouterBench, or imputed from an 8B reference.

model	type	input	output	source
DeepHermes-3-Llama-3-8B-Preview	local	0.0600	0.0900	Imputed
DeepSeek-R1-0528-Qwen3-8B	local	0.0600	0.0900	Imputed
DeepSeek-R1-Distill-Qwen-7B	local	0.0525	0.0788	Imputed
Fin-R1	local	0.0525	0.0788	Imputed
GLM-Z1-9B-0414	local	0.0675	0.1013	Imputed
Intern-S1-mini	local	0.0600	0.0900	Imputed
Llama-3.1-8B-Instruct	local	0.0600	0.0900	Imputed
Llama-3.1-8B-UltraMedical	local	0.0600	0.0900	Imputed
Llama-3.1-Nemotron-Nano-8B-v1	local	0.0600	0.0900	Imputed
MiMo-7B-RL-0530	local	0.0525	0.0788	Imputed
MiniCPM4.1-8B	local	0.0600	0.0900	Imputed
NVIDIA-Nemotron-Nano-9B-v2	local	0.0675	0.1013	Imputed
OpenThinker3-7B	local	0.0525	0.0788	Imputed
Qwen2.5-Coder-7B-Instruct	local	0.0525	0.0788	Imputed
Qwen3-8B	local	0.0600	0.0900	Imputed
claude-opus-4.1	flagship	15.0000	75.0000	Recovered
claude-sonnet-4	flagship	3.0000	15.0000	Recovered
cogito-v1-preview-llama-8B	local	0.0600	0.0900	Imputed
deepseek-r1-0528	flagship	0.5000	2.1500	Recovered
deepseek-v3-0324	flagship	0.2500	0.8800	Recovered
deepseek-v3.1-terminus	flagship	0.2700	1.0000	Recovered
gemini-2.5-flash	flagship	0.3000	2.5000	Recovered
gemini-2.5-pro	flagship	1.2500	10.0000	Recovered
gemma-2-9b-it	local	0.0675	0.1013	Imputed
glm-4-9b-chat	local	0.0675	0.1013	Imputed
glm-4.6	flagship	0.6000	2.2000	Adopted
glm-4.6-3888	flagship	0.6000	2.2000	Recovered
gpt-4.1	flagship	2.0000	8.0000	Recovered
gpt-5	flagship	1.2500	10.0000	Recovered
gpt-5-chat	flagship	1.2500	10.0000	Recovered
granite-3.3-8b-instruct	local	0.0600	0.0900	Imputed
intern-s1	flagship	0.1800	0.5400	Recovered
intern-s1-new	flagship	0.1800	0.5400	Recovered
internlm3-8b-instruct	local	0.0600	0.0900	Imputed
kimi-k2-0905	flagship	0.5000	2.0000	Recovered
qwen3-235b-a22b-2507	flagship	0.0900	0.5800	Recovered
qwen3-235b-a22b-no-thinking	flagship	0.1800	0.5400	Recovered
qwen3-235b-a22b-thinking	flagship	0.1800	0.5400	Recovered
qwen3-235b-a22b-thinking-2507	flagship	0.3000	2.9000	Recovered

(0.10USD/0.15USD) per million input/output tokens. For tau2 and SWE-bench, we retain the reported total costs because the available token counts do not fully explain their multi-turn and reasoning costs. Table 3 lists the token prices and labels them as recovered, adopted, or imputed.

After reconstructing costs, we independently shuffle each task using five random seeds and split the queries into 60% training, 10% validation, and 30% test data. To make cost preferences comparable across tasks, we compute a task- and split-specific scaling factor from the full-pool training frontier, $\text{scale} = \frac{\Delta q}{\Delta c}$, where Δq and Δc denote its attainable quality and cost spans. We evaluate $t \in \{0, 0.5, 1, 2, 4, 8, 16, 32\}$, $\lambda_t = t \cdot \text{scale}$. Thus, $t = 0$ corresponds to quality-only optimization, while larger t places more weight on inference cost.

A.2 Pruning strategies and configurations

The ten pruning strategies comprise 18 configurations: 17 non-random variants and one random baseline averaged over seeds. For in-task pruning, all pruning decisions use training data only. The `single` configuration fits separately for each λ, whereas `grid` averages the corresponding objective over all eight λ values, using the scale of the respective task and training split.

- **Random** removes models in a seeded random order.
- **cheapest-first** removes the model with the highest mean cost first, retaining the cheapest models.
- **highest-avg-quality** removes the model with the lowest mean quality first.
- **highest-avg-utility** removes models with the lowest mean training utility first. We evaluate `single` and `grid` configurations.
- **overlap-quality** identifies the queries on which each model's quality exceeds a threshold of 0.1 or 0.5. It ranks models by their average pairwise Jaccard similarity and removes the most overlapping models first. Similarities are computed once on the full pool. The two thresholds yield two configurations.
- **overlap-conditional-win-loss** measures overlap in the queries where models achieve near-best utility or utility at or below the median. It averages the Jaccard similarities of these win and loss sets. We use three tolerance values, $\varepsilon \in \{10^{-12}, 10^{-4}, 10^{-2}\}$, each with `single` and `grid` variants, yielding six configurations.
- **greedy-regret** repeatedly removes the model whose deletion causes the smallest decrease in the current pool's oracle utility. We evaluate `single` and `grid` configurations.
- **greedy-regret-feasible** uses grid-averaged regret while trying to preserve the pool's minimum achievable training cost, with tolerance 10^{-12}. If every possible removal increases this minimum cost beyond the tolerance, it chooses the removal with the smallest regret.
- **greedy-forward-utility** starts with an empty pool and repeatedly adds the model that gives the largest increase in oracle utility, averaged over the preference grid.
- **frontier-greedy** repeatedly removes the model that leaves the smallest budget gap to the full pool's exact frontier over a fixed training budget interval. Its selection does not depend on the evaluation preference.

A.3 Router configurations

The six router families comprise seven configurations.

- **Best single** always selects the retained model with the highest mean training utility:

$$m^* = \arg\max_{m \in S} \left(q_{m,\text{train}} - \lambda \, c_{m,\text{train}} \right),$$

  where S is the retained model pool.
- **Random router** selects a model uniformly at random from the retained pool.
- **KNN** selects similar training queries using cosine similarity between query embeddings. Their model utilities are weighted by softmax-normalized similarities to predict the best model for a new query. The number of neighbors is selected by validation utility from $\{1, 5, 10, 20, 30, 50, 100, 200\}$, capped at the number of training queries.
- **MF** predicts model scores from a query embedding $\mathbf{e}(x)$:

$$\mathbf{s}(x) = \mathbf{e}(x)W + \mathbf{b},$$

  where W and $\mathbf{b}$ are learned from training data. In `utility` mode, the scores predict utility directly. In `decision` mode, MF predicts quality and selects the model with the highest predicted quality minus λ times its average training cost. This is a multi-model adaptation rather than the original binary RouteLLM (Ong et al., 2025) router.
- **RouterDC** ((Chen et al., 2024)) uses the LLMRouterBench implementation, trained on utility for each λ. We retain the checkpoint with the highest validation utility.

- **GraphRouter** ((Feng et al., 2025)) uses the LLMRouterBench implementation through our repository adapter. It builds a graph containing train, val, and test query embeddings, while withholding test labels. We retain the checkpoint with the highest validation utility.

All query-dependent routers use 1024-dimensional embeddings from `BAAI/bge-m3`, loaded through Sentence Transformers with a maximum input length of 8192 tokens. GraphRouter additionally uses embeddings of task and model descriptions. Model fitting and hyperparameter selection are repeated for each retained pool.

A.4 Experimental tracks

The pool-selection track evaluates every pruned pool, from each task's full model pool down to a minimum size of $k = 3$. Across five seeded partitions and eight cost preferences, with the random pruner evaluated over 10 random seeds, this produces 613,440 plan entries.

For router realization, we use a reduced pool-size ladder, $k \in \{3, 4, 5, 6, 8, 10, 13, 17, 22, 28\}$, to reduce router training costs. Starting at $k = 3$, each next size is approximately 1.3 times the previous size, rounded to the nearest integer. We also reduce the random pruning baseline from ten seeds to three, decreasing the number of pools for which routers must be trained.

For transfer, we exclude the target task from the source data. To prevent leakage, we also exclude tasks that share queries with the target, such as ArenaHard and its category subsets like ArenaHard-Math or ArenaHard-Coding. This leaves 23 source tasks for ArenaHard targets and 26 for other targets. Each pruner uses all source-task data, including train, val, and test queries, to select a reduced target model pool. Routers are then trained on the target task's training split and evaluated on its test split. To reduce training costs, we train routers only on pools of $k \in \{3, 5, 8\}$ models.

For all tracks, we first average random seeds within each configuration, then splits within each task, and finally tasks with equal weight. When averaging across routers, we treat the two MF configurations separately.

A.5 Additional pool-selection and replaceability results

Figure 1 shows one configuration per pruning strategy. For strategies with multiple configurations, we use greedy-regret and highest-avg-utility in `single` mode, greedy-regret-feasible in `grid` mode, overlap-conditional-win-loss with $\varepsilon = 0.01$ in `single` mode, and overlap-quality with threshold 0.5. Figures 4, 5, and 6 extend Figure 1 to all eight cost preferences and all 18 configurations, showing retention, budget gap, and infeasible share. We first average random seeds, then splits within each task, and finally all 27 tasks with equal weight.

We measure each model's removal cost as the decrease in oracle utility when removing it from the full pool, using training queries. Figure 7 complements Figure 2 with the remaining cost preferences.

A.6 Additional router realization results

Figure 8 shows $end2end$ utility for all 18 pruning configurations and eight cost preferences at $k \in \{3, 4, 5, 6, 8, 10\}$, averaged over six non-random routers and 27 tasks.

For each task, we compute the standard deviation across pruning configurations using router-averaged utility, and across non-random routers using pruning-averaged utility. Figure 9 shows the ratio of these task-averaged standard deviations. At $k = 3$, variation across pruning configurations is approximately 11–18 times larger across the eight cost preferences (this difference narrows for larger pools).

A.7 Additional transfer results

We adapt three pruning strategies for transfer because source tasks have different model pools and some target models lack source data. Highest-avg-utility ranks models by normalized source utility, greedy-forward-utility adds models by their average source utility gain, and greedy-regret ranks models by normalized source removal cost. Models without source support receive an average score.

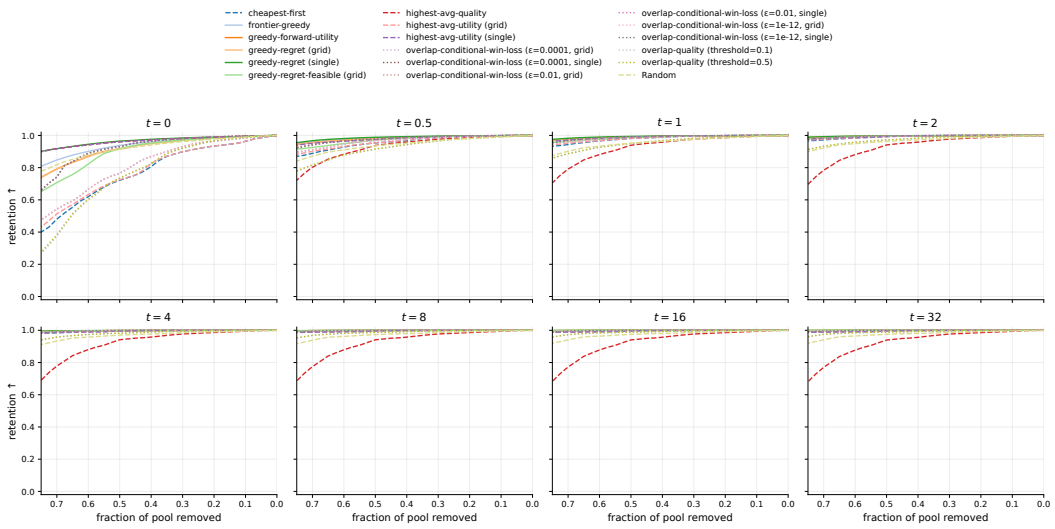

Figure 4: Retention across all pruning configurations and cost preferences.

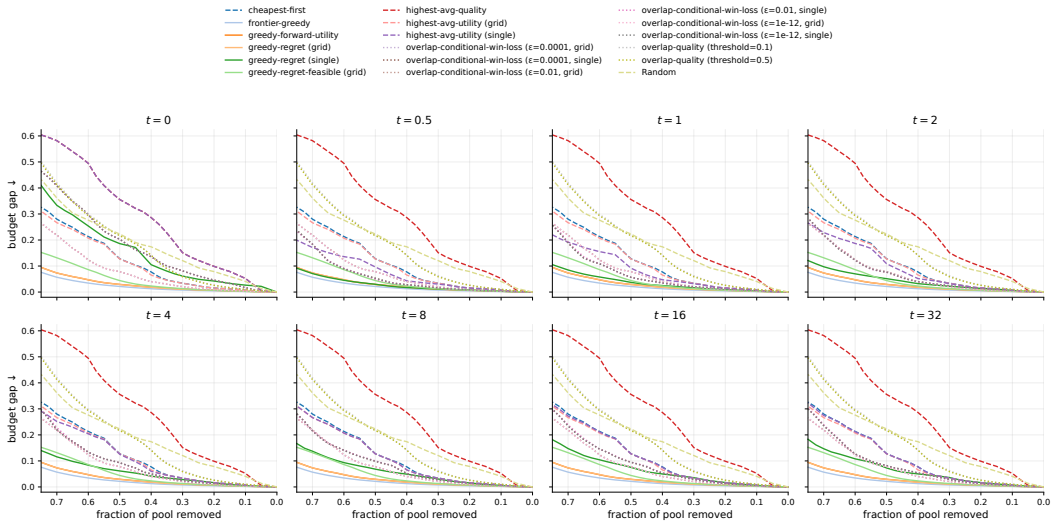

Figure 5: Budget gap across all pruning configurations and cost preferences.

No strategy performs best in every setting. Among these three variants, highest-avg-utility achieves the highest mean $end2end$ utility across all eight cost preferences, three pool sizes ($k \in \{3, 5, 8\}$), and six non-random routers, with tasks weighted equally. We therefore highlight it in the main text. Figures 10 and 11 show all three variants for $k \in \{3, 5, 8\}$ and all eight cost preferences, relative to in-task highest-avg-utility (in-task highest-avg-utility achieves the highest average $end2end$ utility across all pruning configurations).

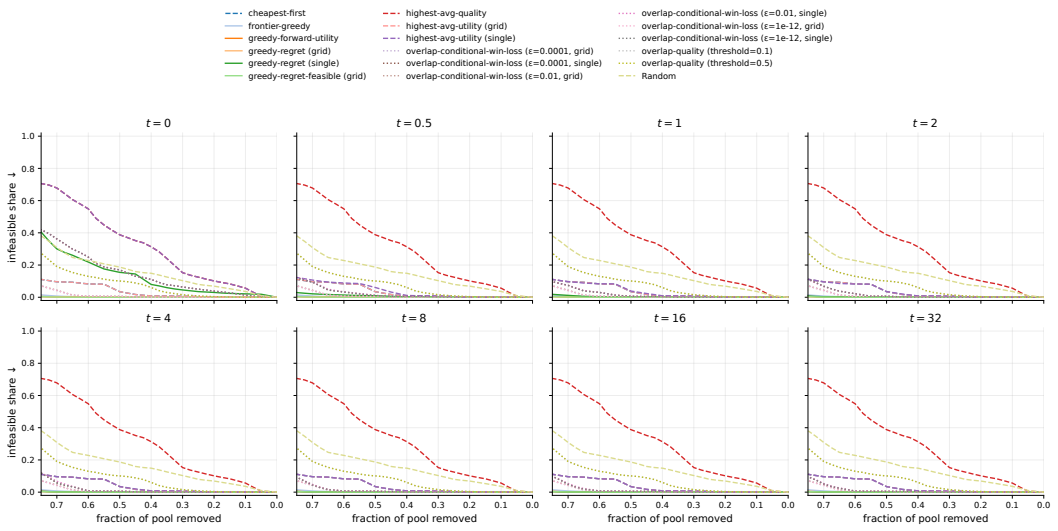

Figure 6: Infeasible share across all pruning configurations and cost preferences.

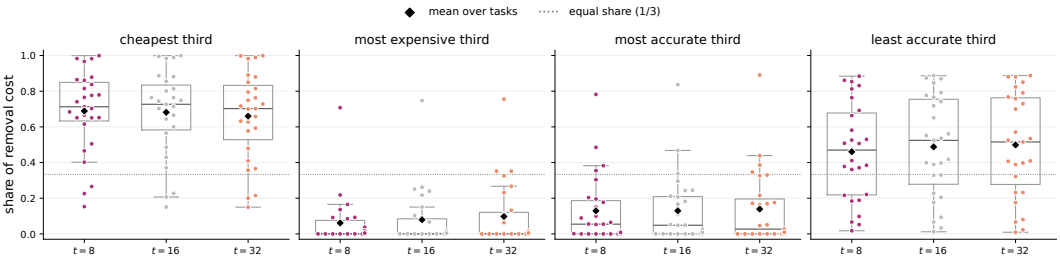

Figure 7: Removal cost shares for the remaining cost preferences $t \in \{8, 16, 32\}$.

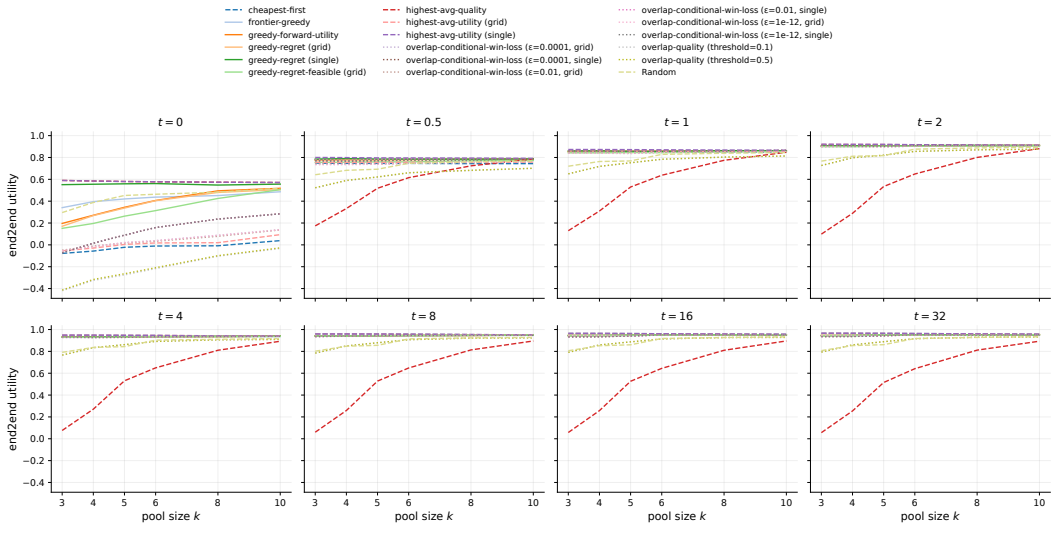

Figure 8: *end2end* utility across all pruning configurations and cost preferences, averaged over six non-random routers and 27 tasks.

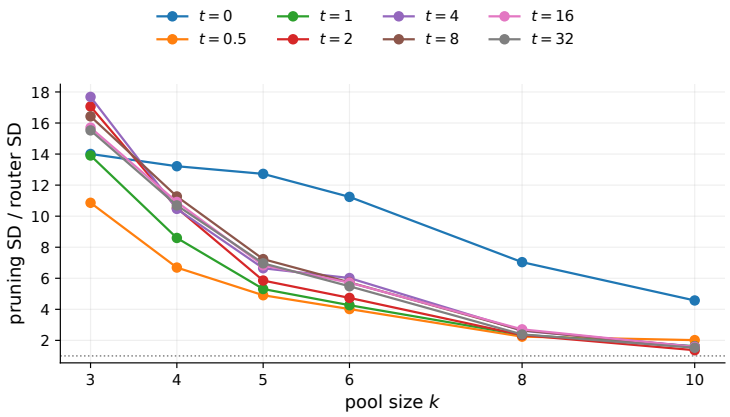

Figure 9: Ratio of task-averaged standard deviations (SD) across pruning configurations and non-random routers. Values above one indicate greater variation across pruning configurations.

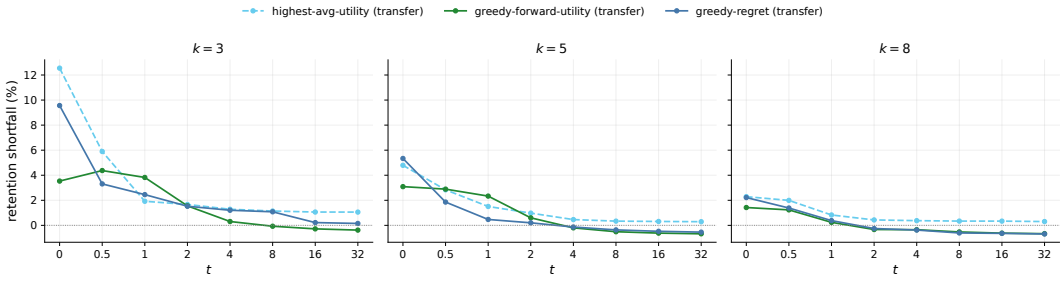

Figure 10: Transfer retention shortfall relative to in-task highest-avg-utility.

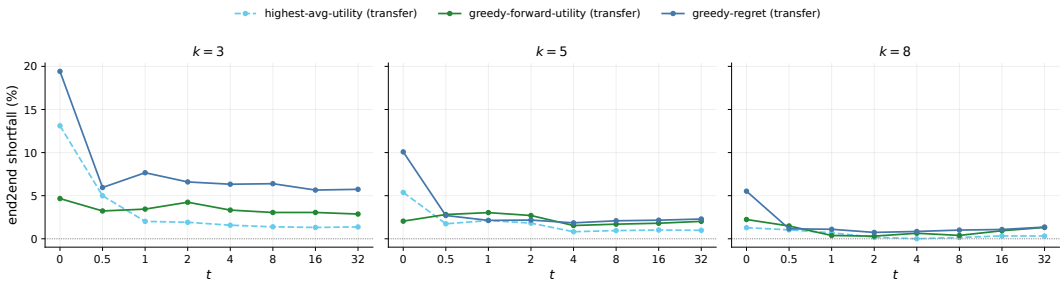

Figure 11: Transfer $end2end$ utility shortfall relative to in-task highest-avg-utility.