RouteLLM: Learning to Route LLMs with Preference Data

Isaac Ong[†]
UC Berkeley
isaacong@berkeley.edu

Amjad Almahairi[†]
Anyscale
anm@anyscale.com

Vincent Wu
UC Berkeley
cveinnt@berkeley.edu

Wei-Lin Chiang
UC Berkeley
weichiang@berkeley.edu

Tianhao Wu
UC Berkeley
thw@berkeley.edu

Joseph E. Gonzalez
UC Berkeley
jegonzal@berkeley.edu

M Waleed Kadous
Canva
waleed@canva.com

Ion Stoica
UC Berkeley & Anyscale
istoica@berkeley.edu

Abstract

Large language models (LLMs) exhibit impressive capabilities across a wide range of tasks, yet the choice of which model to use often involves a trade-off between performance and cost. More powerful models, though effective, come with higher expenses, while less capable models are more cost-effective. To address this dilemma, we propose several efficient router models that dynamically select between a stronger and a weaker LLM during inference, aiming to optimize the balance between cost and response quality. We develop a training framework for these routers leveraging human preference data and data augmentation techniques to enhance performance. Our evaluation on widely-recognized benchmarks shows that our approach significantly reduces costs—by over 2 times in certain cases—without compromising the quality of responses. Interestingly, our router models also demonstrate significant transfer learning capabilities, maintaining their performance even when the strong and weak models are changed at test time. This highlights the potential of these routers to provide a cost-effective yet high-performance solution for deploying LLMs.

1 Introduction

Recent advances in large language models (LLMs) have demonstrated remarkable capabilities across a wide range of natural language tasks. From open-ended conversation and question answering to text summarization and code generation, LLMs have showcased an impressive level of fluency and understanding [1, 8]. This rapid progress has been enabled by a combination of architectural innovations, such as the Transformer architecture [27], as well as improvements in scaling up data and training infrastructure [7, 23].

However, not all LLMs are created equal—there exists wide variation in the costs and sizes of LLMs, which can range in size from one billion to hundreds of billions of parameters. LLMs also differ in terms of the data they are trained on, which in turn leads to variations in the strengths, weaknesses, and capabilities of different models. Broadly speaking, larger models tend to be more capable but come at a higher cost, while smaller models tend to be less capable but cheaper to serve.

[†] Equal contribution.

Preprint. Under review.

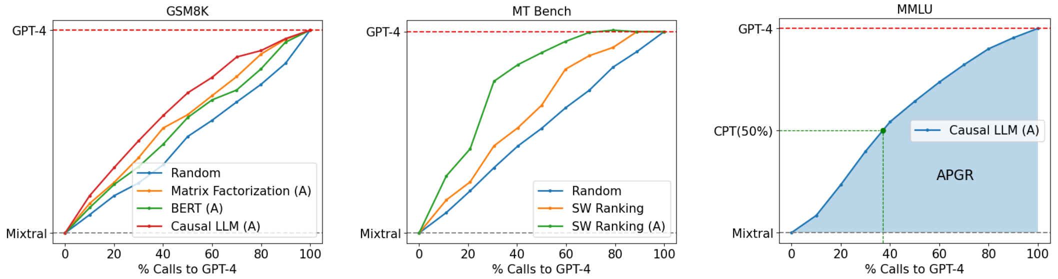

Figure 1: Routing performance/cost trade-off between GPT-4 and Mixtral-8x7B. *(left)* We demonstrate several routers that outperform the random baseline on OOD eval GSM8K. *(center)* We demonstrate improvement in router performance through data augmentation, denoted by (A), on MT Bench. *(right)* We display the main metrics we consider: call-performance threshold (CPT, denoted in green) and average performance gain recovered (APGR, denoted by the blue shaded region).

This heterogeneous landscape presents a dilemma in the practical deployment of LLMs in real-world applications. While routing all user queries to the largest, most capable model ensures high-quality results, it is prohibitively expensive. Conversely, routing queries to smaller models can save costs—up to more than 50x (e.g., Llama-3-70b vs. GPT-4, or Claude-3 Haiku vs. Opus[1])—but may result in lower quality responses, as the smaller model may not be able to handle complex queries effectively.

To this end, *LLM routing* is a promising solution to this problem, whereby each user query is first processed by a *router model*, before deciding which LLM to route the query to. This can potentially route easier queries to smaller models, and more difficult queries to larger models, optimizing the quality of model responses while minimizing cost. However, optimal LLM routing—defined as achieving the highest quality given a cost target or minimizing cost given a quality target—is a challenging problem. A robust router model needs to infer the intent, complexity, and domain of an incoming query as well as understand candidate models' capabilities to route the query to the most appropriate model. Furthermore, the router model needs to be economical, fast, and adaptive to the evolving model landscape, where new models with improved capabilities are continually introduced.

In this work, we present a principled framework for query routing between LLMs. Our setup involves learning to route between a stronger model and a weaker model as seen in Figure 1. Our objective is to minimize costs while achieving a specific performance target, such as 90% of the stronger model's performance, by intelligently routing simpler queries to the weaker model and reserving more complex queries for the stronger model. We develop a training framework for router systems utilizing human preference data and data augmentation techniques. We evaluate our router models on widely recognized benchmarks, such as MMLU [15] and MT Bench [30], and demonstrate that our framework can significantly reduce costs—by over 2 times—without substantially compromising response quality.

To summarize, we make the following contributions:

- We formulate the LLM routing problem to explore the trade-off between cost and response quality.
- We propose a router training framework based on human preference data and augmentation techniques, demonstrating over 2x cost saving on widely used benchmarks.
- We open-source the code and preference data used to train our routers.[2].

Several recent studies have explored optimizing the cost and performance trade-offs in deploying large language models (LLMs). LLM-BLENDER [17] employs an ensemble framework that calls multiple LLMs at inference and uses a router model to select the best response. Frugal-GPT [9] employs an LLM cascade, sequentially querying LLMs until a reliable response is found. Both approaches' inference cost grows with the number of models involved. Our approach, by contrast, routes each query to a single LLM. A closely related study, Hybrid-LLM [13], shares similarities

[1] Per one million output tokens: Llama-3-70b ($1) vs. GPT-4-0613 ($60), Haiku ($1.25) vs. Opus ($75)

[2] https://github.com/lm-sys/RouteLLM

with our framework but differs in three key ways: it uses synthetic preference labels derived via BARTScore [29], relies on a single BERT-based router architecture, and limits evaluation to in-domain generalization. In contrast, our work leverages human preference labels from Chatbot Arena [10], explores several router architectures, and demonstrates that that augmenting the dataset results in significant performance improvements across all router architectures. Additionally, we emphasize out-of-domain generalization by evaluating on multiple public benchmarks.

2 LLM Routing

2.1 Problem Formulation

Consider a set of N different LLM models $\mathcal{M} = \{M_1, \ldots, M_N\}$. Each model $M_i : \mathcal{Q} \to \mathcal{A}$ can be abstracted as a function that maps a query to an answer. A *routing function* $R : \mathcal{Q} \times \mathcal{M}^N \to \{1, \ldots, N\}$ is an N-way classifier that takes a query $q \in \mathcal{Q}$ and selects a model to answer q, with the answer being $a = M_{R(q)}(q)$. The challenge of routing involves achieving an optimal equilibrium between increasing response quality and reducing cost[3]. Assume we have access to *preference data*: $\mathcal{D}_{\text{pref}} = \{(q, l_{i,j}) \mid q \in \mathcal{Q}, i, j \in N, l_{i,j} \in \mathcal{L}\}$, where q is a query, and $l_{i,j}$ is a label representing the comparison outcome of comparing M_i, M_j's quality on q, which takes values in $\mathcal{L} = \{\text{win}_{M_i}, \text{tie}, \text{win}_{M_j}\}$. It is important to distinguish between reward modeling [22] and routing. Reward modeling evaluates response quality post-LLM generation, whereas routing requires the router to select the appropriate model before seeing the response. This necessitates a deep understanding of the question's complexity as well as the strengths and weaknesses of the available LLMs.

In this work, we focus on routing between two classes of models: (1) *strong models* ($\mathcal{M}_{\text{strong}}$), consisting of models capable of producing high quality responses but at a high cost. This class primarily consists of the most advanced closed-source models, such as GPT-4. (2) *weak models* ($\mathcal{M}_{\text{weak}}$), consisting of models with a relatively lower quality and lower cost, such as such as Mixtral-8x7B. This binary routing problem is quite common in practice, particularly as developers of LLM applications strive to balance quality and cost. Furthermore, addressing this problem forms the foundation for solving a more general N-way routing problem.

We present a principled framework for learning a binary routing function $R^{\alpha}_{\text{bin}} : \mathcal{Q} \to \{0, 1\}$ between $\mathcal{M}_{\text{weak}}$ and $\mathcal{M}_{\text{strong}}$ from preference data. To achieve this, we define R^{α}_{bin} using two components:

1) **Win Prediction Model** which predicts the probability of winning for strong models $\mathcal{M}_{\text{strong}}$, i.e. $P_{\boldsymbol{\theta}}(\text{win}_{\mathcal{M}_{\text{strong}}} | q)$. In our binary classification setting, this probability captures the win/loss probability of both model classes.[4] We can learn the parameters of this model $\boldsymbol{\theta}$ with maximum likelihood on the preference data:

$$\max_{\boldsymbol{\theta}} \sum_{(q, l_{i,j}) \in \mathcal{D}_{\text{pref}}} \log P_{\boldsymbol{\theta}}(l_{i,j} \mid q). \tag{1}$$

By learning the winning probability on preference data, we capture the strengths and weaknesses of both model classes on various kinds of queries. In Section 3.2, we propose several approaches for parameterizing the win prediction model.

2) **Cost Threshold** $\alpha \in [0, 1]$ which converts the winning probability into a routing decision between $\mathcal{M}_{\text{strong}}$ and $\mathcal{M}_{\text{weak}}$. Given a query q, the routing decision is formulated as:

$$R^{\alpha}_{\text{bin}}(q) = \begin{cases} 0 \text{ (i.e., } \mathcal{M}_{\text{weak}}) & \text{if } P(\text{win}_{M_j} \mid q) < \alpha, \\ 1 \text{ (i.e., } \mathcal{M}_{\text{strong}}) & \text{otherwise.} \end{cases} \tag{2}$$

The threshold α controls the quality/cost trade-off: a higher threshold imposes a stricter cost constraint, reducing expenses but potentially compromising quality.

Finally, we denote the *router's response* as $\mathcal{M}_{R^{\alpha}_{\text{bin}}(q)}(q)$, which represents the response generated by either the weak or strong model, depending on the router's decision.

[3] We focus on inference cost in \$ cost/token, but the same concept applies to other costs such as latency.

[4] We consider "tie" as a win for the weaker models.

2.2 Metrics

In this section, we define evaluation metrics that capture the trade-off between cost and quality in the LLM routing problem. We start with metrics that independently assess the quality and cost efficiency of a given R^{α}_{bin}, then introduce two compounded metrics which we use in our experimental evaluations.

For cost efficiency, we calculate the *percentage of calls to the strong model*:

$$c(R^{\alpha}_{\text{bin}}) = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathbb{I}\{R^{\alpha}_{\text{bin}}(q) = 1\}, \tag{3}$$

since $\mathcal{M}_{\text{strong}}$ models incur significantly higher costs than $\mathcal{M}_{\text{weak}}$ models.

For quality, we measure the *average response quality* on an evaluation set $\mathcal{Q}$:

$$r(R^{\alpha}_{\text{bin}}) = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \delta(\mathcal{M}_{R^{\alpha}_{\text{bin}}(q)}(q)), \tag{4}$$

where $\delta(\mathcal{M}_{R^{\alpha}_{\text{bin}}(q)}(q))$ represents a numerical score of the router's response to q. This score can be result of a predefined metric measuring the correctness of the response in golden-labeled datasets (e.g., MMLU) or a numerical label (e.g., 1-5 or 1-10) where higher values indicate better quality.

Given that the performance of R^{α}_{bin} lies between the weak and strong models' performance, we quantify the router's performance relative to the performance gap between both models. We define the overall performance gain of R^{α}_{bin} with the *performance gap recovered (PGR)*:

$$PGR(R^{\alpha}_{\text{bin}}) = \frac{r(R^{\alpha}_{\text{bin}}) - r(\mathcal{M}_{\text{weak}})}{r(\mathcal{M}_{\text{strong}}) - r(\mathcal{M}_{\text{weak}})}. \tag{5}$$

Neither of these metrics alone is sufficient, as they do not capture the quality-cost trade-off in routing. For instance, a trivial router that sends every query to the strong model achieves a perfect $PGR = 1$ without any cost reduction. Therefore, we compute a *call-performance graph* for a router R_{bin} by varying the threshold values α. We define the **average performance gap recovered (APGR)** as an overall measure of how well the router can recover the performance gap under different cost constraints:

$$APGR(R_{\text{bin}}) = \int_0^1 PGR(R^{\alpha}_{\text{bin}})\, d\left(c(R^{\alpha}_{\text{bin}})\right). \tag{6}$$

In Figure 1-*(right)*, APGR is represented by the area between the router's performance curve and the weak model's performance. Empirically, we discretize the percentage of calls interval $[0\%, 100\%]$ into $\{c_i\}_{i \in [10]}$. For each c_i, we determine the cutoff threshold α_i that meets the cost constraint. We approximate $APGR$ using the following formula:

$$APGR(R_{\text{bin}}) \approx \frac{1}{10} \sum_{i=1}^{10} PGR(R^{\alpha_i}_{\text{bin}}) \tag{7}$$

In many real-world applications, it is important to quantify the cost required to achieve a certain level of performance. Therefore, we define a second metric which we call **call-performance threshold (CPT)**. Given a desired router performance (measured as PGR of $x\%$), the CPT($x\%$) refers to the *minimum percentage* of calls to the strong model required to obtained the desired PGR. In Figure 1-*(right)*, the dotted green line denotes CPT(50%), i.e. the percentage of calls to GPT-4 required to achieve the desired performance of 50% PGR; here, $CPT(50\%) \approx 37\%$.

3 Methodology

3.1 Preference Data

We start by describing how we can obtain the preference data to train the routing function. We primarily use 80k battles from the online Chatbot Arena platform [10]. On this platform, users interact with a chatbot interface and submit prompts of their choice. Upon submission, they receive responses from two anonymous models and vote for a winning model or a tie. The resulting dataset, denoted as $\mathcal{D}_{\text{arena}} = \{(q, a_i, a_j, l_{i,j}) \mid q \in \mathcal{Q}, a_i, a_j \in \mathcal{A}, l_{i,j} \in \mathcal{L}\}$, consisting of user queries, answers from two models M_i, M_j, and a pairwise comparison label based on human judgment.

A major issue with using the raw Chatbot Arena data is label sparsity. For instance, the percentage of comparison labels between any two models, on average, is less than 0.1%. Therefore, we derive the preference data for training the router as follows: First, we reduce label sparsity by clustering models in $\mathcal{D}_{\text{arena}}$ into 10 different tiers (see Appendix A), using each model's Elo score on the Chatbot Arena leaderboard[5] and aiming to minimize the variation within each tier via dynamic programming. We choose the models on the first and second tiers to represent strong models $\mathcal{M}_{\text{strong}}$, and the models on the third tier to represent weak models $\mathcal{M}_{\text{weak}}$. While we primarily train on battles across these tiers, we also leverage battles involving other model tiers to regularize our learning methods. Crucially, we omit the actual model responses in $\mathcal{D}_{\text{arena}}$, retaining only the model identities, i.e. $e \sim \mathcal{D}_{\text{pref}}$ is $e = (q, M_i, M_j, l_{i,j})$. The comparison label $l_{i,j}$ still provides insight into the relative capabilities of the LLMs M_i and M_j on various types and complexity levels of the query q.

3.1.1 Data Augmentation

Even after classifying models into tiers, the human preference signal can still be quite sparse across different model classes. As we discuss in Sec 4.1, we find that this can hinder generalization, especially for parameter-heavy models. Thus we explore two data augmentation methods:

Golden-labeled datasets: We augment our training data with datasets of the form $\mathcal{D}_{\text{gold}} = \{(q, a, l_g) \mid q \in \mathcal{Q}, a \in \mathcal{A}, l_g \in \mathbb{R}\}$, where a golden label l_g is automatically calculated for a model answer a, e.g. multiple-choice answers. One example of such a dataset is the MMLU benchmark [15]. We use the validation split of MMLU containing approximately 1500 questions and derive comparison labels $l_{i,j}$ from l_g by simply comparing the responses from M_i and M_j, creating the preference dataset $\mathcal{D}_{\text{gold}}$ for augmentation.

LLM-judge-labeled datasets: We explore obtaining preference labels on open-ended purpose chat domains using a LLM judge [30], as it has demonstrated a high correlation with human judgment [14, 17]. Given a collection of user queries, we start by generating responses from both a strong model in $\mathcal{M}_{\text{strong}}$ and a weak model in $\mathcal{M}_{\text{weak}}$ and then produce pairwise comparison labels using GPT-4 as a judge. The primary challenge with this method is the high cost of collecting responses and pairwise comparisons from GPT-4 in large quantities. Fortunately, the Nectar dataset [31] offers a wide variety of queries with corresponding model responses. We significantly reduce our costs by selecting queries with GPT-4 responses (as a representative of $\mathcal{M}_{\text{strong}}$), on which we generate responses from Mixtral-8x7B (as representative of $\mathcal{M}_{\text{weak}}$). Finally, we obtain pairwise comparison labels using the GPT-4 judge.[6]. Overall, we collect a preference dataset $\mathcal{D}_{\text{judge}}$ of approximately 120K samples with a total cost of around \$700 USD.

3.2 Routing Approaches

We now detail our methods for learning the win prediction model $P_{\boldsymbol{\theta}}(\text{win}_{\mathcal{M}_{\text{strong}}}|q)$ from preference data $\mathcal{D}_{\text{pref}}$. We denote a sample $(q, M_i, M_j, l_{i,j}) \sim \mathcal{D}_{\text{pref}}$ as $e = (q, M_w, M_l)$, where M_w and M_l refer to the winning and losing model respectively.

Similarity-weighted (SW) ranking We adopt a Bradley-Terry (BT) model [6] similar to [10]. Given a user query q, we compute a weight $\omega_i = \gamma^{1+\mathcal{S}(q,\hat{q})}$ [7] for each query q_i in the train set based on its similarity to q:

$$\mathcal{S}(q, q_i) = \frac{\epsilon \cdot \epsilon_i}{\|\epsilon\|\|\epsilon_i\| \cdot \max_{1 \leq s \leq |\mathcal{D}_{\text{pref}}|} \frac{\epsilon_i \cdot \epsilon_s}{\|\epsilon_i\|\|\epsilon_s\|}}, \tag{8}$$

where ϵ denotes a query embedding. We learn BT coefficients ξ (representing 10 model classes) by solving:

$$\underset{\xi}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{D}_{\text{pref}}|} \left[\omega_i \cdot \ell \left(l_i, \frac{1}{1 + e^{\xi_{w_i} - \xi_{l_i}}} \right) \right], \tag{9}$$

[5] https://leaderboard.lmsys.org

[6] We employ best practices recommended in [30] to de-bias GPT-4 judgements

[7] We find that exponential scale works best in practice and choose $\gamma = 10$

where ℓ is a binary cross-entropy loss. The resulting BT coefficients allow us to estimate the win probability as: $P(\text{win}_{M_w}|q) = \frac{1}{1+e^{\xi_w - \xi_l}}$. For this router model, there is no training required and the solving is performed at inference time.

Matrix factorization Inspired by matrix factorization models [18, 25] in recommendation systems to capture the low-rank structure of user-item interactions, we leverage this approach for training on preference data. The key is to uncover a hidden scoring function $s : \mathcal{M} \times \mathcal{Q} \to \mathbb{R}$. The score $s(M_w, q)$ should represent the quality of the model M_w's answer to the query q, i.e. if a model M_w is better than M_l on a query q, then $s(M_w, q) > s(M_l, q)$. We enforce this relationship by modeling the win probability with a BT relationship [6]:

$$P(\text{win}_{M_w}|q) = \sigma(s(M_w, q) - s(M_l, q)), \tag{10}$$

which we optimize on preference data. We model the scoring function s as a bilinear function of model and query, and embed the model identity M to a d_m-dimensional vector v_m and the query to a d_q-dimensional vector v_q:

$$s(M, q) = w_2^T(v_m \odot (W_1^T v_q + b)) \tag{11}$$

Here, $\odot$ represents the Hadamard product, $W_1 \in \mathbb{R}^{d_q \times d_m}$ and $b \in \mathbb{R}^{d_m}$ is the projection layer to align the dimension of v_q with v_m, and $w_2 \in \mathbb{R}^{d_m}$ is the linear regression layer to produce the final scalar. This method is essentially learning a matrix factorization of the score matrix on the set $\mathcal{Q} \times \mathcal{M}$. We train the model on a 8GB GPU for ≈ 10 epochs, using batch size 64 and the Adam optimizer with learning rate 3×10^{-4} and weight decay 1×10^{-5}.

BERT classifier Here we explore using a standard text classification method with a higher number of parameters compared to previous methods. We use a BERT-base architecture [12], to give a contextualized embedding of the user query, and define win probability as:

$$P_{\boldsymbol{\theta}}(\text{win}_{M_w}|q) = \sigma(Wh_{\text{CLS}} + b), \tag{12}$$

where h_{CLS} is an embedding corresponding to the special classification token (CLS) summarizing the input query q; and W, b, σ are parameters and sigmoid activation of a logistic regression head. We perform full-parameter fine-tuning on $\mathcal{D}_{\text{pref}}$. We train the model on 2xL4 24GB GPUs for ~ 2000 steps using a batch size of 16, maximum sequence length of 512, learning rate of 1×10^{-5} and weight decay of 0.01.

Causal LLM classifier We finally expand the capacity of our router by parameterizing it with Llama 3 8B[2]. We use an instruction-following paradigm [28], i.e. we provide as input an instruction prompt containing the user query, and output the win probability in a next-token prediction fashion – instead of using a separate classification head. Notably, we append the comparison labels as additional tokens to the vocabulary, and compute the win probability as softmax over the label classes $\mathcal{L}$. We train the model on 8xA100 80GB GPUs for ~ 2000 steps using a batch size of 8, maximum sequence length of 2048, and a learning rate of 1×10^{-6}.

4 Experiments

Training data: As mentioned in Sec. 3.1, we primarily use the 80K Chatbot Arena for training our models, but hold out 5k samples for validation. We prune all prompt samples shorter than 16 characters, resulting in 65k pairwise comparisons between 64 different models. These consist of conversations from over 100 languages, with the bulk of the conversations (81%) in English, followed by Chinese (3.1%), and Russian (2.2%). We assign models to 10 classes to reduce sparsity of comparison labels. As discussed in Sec. 3.1.1, we further augment our training data with with either: 1) $\mathcal{D}_{\text{gold}}$, golden-labeled data created from the MMLU validation split and 2) $\mathcal{D}_{\text{judge}}$, GPT-4-as-a-judge labeled chat data.

Evaluation benchmarks: We evaluate our routers on three widely-used academic benchmarks: MMLU [15] consisting of 14,042 questions across 57 subjects, MT Bench [30] with 160 open-ended questions using LLM-as-a-judge, and GSM8K [11] with over 1,000 grade school math problems. Additionally, we conduct a cross-contamination check between our evaluation and training datasets, and report uncontaminated results below. We present results on public benchmarks to understand the out-of-domain generalization of routers.

Routers: For both the matrix factorization router and the similarity-weighted ranking router, we use OpenAI's embedding model `text-embedding-3-small` to embed the input query. We perform full-parameter finetuning on both BERT and Causal LLM, and use the validation set for model selection. We opt to use `gpt-4-1106-preview` [20] as a representative model in $\mathcal{M}_{\text{strong}}$, and Mixtral 8x7B [16] as a representative model in $\mathcal{M}_{\text{weak}}$, to concretely evaluate router performance. We also use a random router that routes queries randomly under a cost constraint as a baseline.

4.1 Results

Training data	Method	$CPT(50\%)$	$CPT(80\%)$	$APGR$	
	Random (95% CI)	49.03(±4)%	78.08(±3)%	0.500(±0.02)	(+0%)
$\mathcal{D}_{\text{arena}}$	BERT	78.09%	87.64%	0.391	(-21.8%)
	Causal LLM	28.82%	77.53%	0.573	(+14.6%)
	Matrix Factorization	**25.32%**	74.26%	0.580	(+16%)
	SW Ranking	37.85%	**58.99%**	**0.610**	(+22.1%)
$\mathcal{D}_{\text{arena}} + \mathcal{D}_{\text{judge}}$	BERT	19.58%	34.02%	0.751	(+50.2%)
	Causal LLM	31.50%	48.75%	0.679	(+35.8%)
	Matrix Factorization	**13.40%**	**31.31%**	**0.802**	(+60.4%)
	SW Ranking	23.21%	36.04%	0.759	(+51.8%)

Table 1: MT Bench results. Note that the score for CPT at 50% (8.8) is 95% that of GPT-4 performance (9.3). Our routers exhibit strong performance on MT Bench when trained on $\mathcal{D}_{\text{arena}}$, with further improvement when the dataset is augmented with $\mathcal{D}_{\text{judge}}$, reducing costs by up to 75% as compared to the random router.

Table 1 displays our router performance on MT Bench. For routers trained on the Arena dataset, we observe strong performance for both matrix factorization and similarity-weighted ranking, with both routers performing significantly better than the random router across all metrics. Notably, matrix factorization requires half the number of GPT-4 calls as compared to random to achieve a PGR of 50%. However, our BERT and causal LLM classifiers perform close to random when trained on the Arena dataset, which we attribute to high capacity approaches performing worse in a low-data regime.

Augmenting the preference data using a GPT-4 judge leads to notable improvements across all routers. The BERT and causal LLM routers now perform much better than the random baseline, with the BERT classifier achieving an APGR improvement of over 50% as compared to random. When trained on this augmented dataset, matrix factorization is the best-performing router as its CPT(80%) is nearly halved, requiring 50% less GPT-4 calls as compared to random.

We also compare the MT Bench performance of our routers against existing routing systems in Appendix E, demonstrating the substantial improvements that our routers achieve over other available systems.

Training data	Method	$CPT(50\%)$	$CPT(80\%)$	$APGR$	
	Random (95% CI)	50.07(±0)%	79.93(±0)%	0.500(±0)	(+0%)
$\mathcal{D}_{\text{arena}}$	BERT	49.43%	77.80%	0.502	(+0.5%)
	Causal LLM	48.88%	77.93%	0.499	(-0.2%)
	Matrix Factorization	**45.00%**	**76.86%**	**0.524**	(+4.9%)
	SW Ranking	55.82%	80.25%	0.473	(-5.4%)
$\mathcal{D}_{\text{arena}} + \mathcal{D}_{\text{gold}}$	BERT	41.30%	72.20%	0.572	(+14.4%)
	Causal LLM	35.49%	**70.31%**	0.600	(+19.9%)
	Matrix Factorization	35.46%	71.40%	0.597	(+19.5%)
	SW Ranking	**35.40%**	71.55%	**0.603**	(+20.7%)

Table 2: 5-shot MMLU results for our routers. Note that the score for CPT at 50% (75) is 92% that of GPT-4 performance (81). Routers trained only on $\mathcal{D}_{\text{arena}}$ perform poorly due to most questions being out-of-distribution, but dataset augmentation with $\mathcal{D}_{\text{gold}}$ is highly effective, leading to significant improvement in router performance even with a small number of samples.

On MMLU (Table 2), all routers perform poorly at the level of the random router when trained only on Arena dataset, which we attribute to most MMLU questions being out-of-distribution (see Section 4.2). However, augmenting the training dataset with golden-label data from the MMLU validation split leads to significant performance improvements on MMLU across all routers, with all routers requiring approximately 20% less GPT-4 calls than random for CPT(50%). Importantly, this is despite the fact that the additional golden-labeled dataset of approximately 1500 samples represents less than 2% of the overall training data, demonstrating the effectiveness of dataset augmentation even when the number of samples is small.

Training data	Method	CPT(50%)	CPT(80%)	$APGR$	
	Random (95% CI)	50.00(±2)%	80.08(±1)%	0.497(±0.01)	(+0%)
$\mathcal{D}_{\text{arena}}$	BERT	58.78%	83.84%	0.438	(-11.8%)
	Causal LLM	56.09%	83.56%	0.461	(-7.3%)
	Matrix Factorization	**53.59%**	85.24%	0.4746	(-4.5%)
	SW Ranking	54.43%	**82.11%**	**0.4753**	(-4.3%)
$\mathcal{D}_{\text{arena}} + \mathcal{D}_{\text{judge}}$	BERT	44.76%	79.09%	0.531	(+6.9%)
	Causal LLM	**33.64%**	**63.26%**	**0.622**	(+25.3%)
	Matrix Factorization	38.82%	72.62%	0.565	(+13.8%)
	SW Ranking	41.21%	72.20%	0.568	(+14.3%)

Table 3: 8-shot GSM8K results. Note that the score for CPT at 50% (75) is 87% that of GPT-4 performance (86). Routers trained only on $\mathcal{D}_{\text{arena}}$ again perform poorly due to questions being out-of-distribution, but augmentation with $\mathcal{D}_{\text{judge}}$ substantially improves router performance.

Finally, on GSM8K (Table 3), we observe that similar to MMLU, the performance of all routers trained only on the Arena dataset is close to random. However, training our routers on the dataset augmented with synthetic data from an LLM judge improves performance substantially, with all routers going from an APGR worse than random to an APGR greater than random. When trained on this augmented dataset, the causal LLM classifier performs the best out of all routers, requiring 17% less GPT-4 calls than random to achieve CPT(50%) and CPT(80%).

4.2 Quantifying dataset and benchmark similarity

We attribute the difference in the performance of routers trained on the same dataset across different benchmarks to the differing distributions of evaluation data and training data. For each benchmark-dataset pair, we compute a *benchmark-dataset similarity score* in Table 4 indicating how well-represented evaluation data is in the training data, described in detail in Appendix C.

	Arena	Arena augmented with	Arena augmented
		with $\mathcal{D}_{\text{judge}}$	with $\mathcal{D}_{\text{gold}}$
MT Bench	0.6078	0.6525	-
MMLU	0.4823	-	0.5678
GSM8K	0.4926	0.5335	-

Table 4: Benchmark-dataset similarity scores demonstrate a strong correlation between these scores and the performance of routers on the corresponding benchmarks, providing a way of quantitatively improving router performance.

A higher benchmark-dataset similarity score is correlated with stronger performance on that benchmark for routers trained using the corresponding dataset, as shown in Section 4.1. Dataset augmentation, be it using golden-labeled datasets or LLM-judge-labeled datasets, shifts the overall distribution of the preference data to be more in line with the benchmarks and increases the benchmark-dataset similarity score, which translates into performance improvements. This similarity score is also useful for understanding the relative performance of routers across different benchmarks: on the Arena dataset, the similarity score between MT bench and all datasets is noticeably greater than other benchmarks, which we believe explains the relatively stronger router performance on MT Bench as compared to GSM8K and MMLU. Benchmark-dataset similarity scores are a promising direction to systematically improve router performance in real-world use cases given knowledge about the query distribution.

4.3 Generalizing to other model pairs

We pick `gpt-4-1106-preview` [20] and Mixtral 8x7B [16] as representative strong and weak models for the above experiments. However, to demonstrate the generalizability of our framework to different model pairs, we report in this section our router performance on MT Bench when routed between Claude 3 Opus [5] and Llama 3 8B [4]. Importantly, we use the same routers *without any retraining*, and only replace the strong model and weak model routed to. These two models are also not present in our training data.

Train Set	Method	$CPT(50\%)$	$CPT(80\%)$	$APGR$	
	Random (95% CI)	47.23(±5)%	77.08(±5)%	0.494(±0.03)	(+0%)
$\mathcal{D}_{\text{arena}}$	BERT	60.61%	88.39%	0.475	(-3.9%)
	Causal LLM	31.96%	**59.83%**	**0.645**	(+30.5%)
	Matrix Factorization	**24.84%**	85.73%	0.605	(+22.5%)
	SW Ranking	33.54%	80.31%	0.553	(+12%)
$\mathcal{D}_{\text{arena}} + \mathcal{D}_{\text{judge}}$	BERT	36.28%	50.83%	0.618	(+25.2%)
	Causal LLM	40.46%	55.83%	0.625	(+26.5%)
	Matrix Factorization	**30.48%**	**41.81%**	**0.703**	(+42.2%)
	SW Ranking	31.67%	48.39%	0.667	(+35%)

Table 5: MT Bench results for Claude 3 Opus / Llama 3 8B. Our routers generalize very well across different strong and weak model pairs without any retraining.

Again, we observe strong results across all existing routers on MT Bench even when the model pair is replaced. Performance across all routers is comparable to with the original model pair. Results for both the new model pair and original model pair are still significantly stronger than random, with our routers requiring up to 30% less GPT-4 calls than random to achieve CPT(80%). These results suggest that our routers have learned some common characteristics of problems that can distinguish between strong and weak models, which generalize to new strong and weak model pairs without additional training.

4.4 Cost analysis

	CPT 50%	CPT 80%
MT Bench	3.66x (at 95% GPT-4 quality)	2.49x
MMLU	1.41x (at 92% GPT-4 quality)	1.14x
GSM8K	1.49x (at 87% GPT-4 quality)	1.27x

Table 6: Cost saving ratio of our best performing routers over GPT-4. Our routers are able to achieve significant cost savings while maintaining quality.

We estimate the average cost of using GPT-4 and Mixtral 8x7B to be $24.7 per million tokens and $0.24 per million tokens respectively (see details in Appendix D). In Table 6, we show results of quantifying the cost savings achieved by our approach. We calculate the inverse of the ratio of GPT-4 calls utilized by our top-performing router relative to the random baseline because the cost of GPT-4 is the dominant factor in our analysis. Our routers achieve optimal cost savings of up to 3.66x, demonstrating that routing can significantly reduce cost while maintaining response quality.

4.5 Routing Overhead

	Cost / million requests	Requests / second	Hourly cost of VM
SW Ranking	$37.36	2.9	$0.39
Matrix Factorization	$1.42	155.16	$0.8
BERT	$3.19	69.62	$0.8
Causal LLM	$5.23	42.46	$0.8

Table 7: Cost and inference overhead of different routers. As compared to the cost of LLM generation, the cost of deploying a router is small while also able being able to support real-world workloads.

A concern with LLM routing is the overhead of routing as compared to using a single model. Therefore, we measure and report the overhead of our routers in Table 7 to demonstrate their practicality using randomly-sampled conversations from Chatbot Arena. For routers that requires GPUs, namely matrix factorization and the classifier methods, we utilize Google Cloud's `g2-standard-4` VM containing a single NVIDIA L4 GPU. For similarity-weighted ranking, we use Google Cloud's CPU-only `n2-standard-8` VM. Our GPU-based routers are currently much more efficient that our CPU-based routers, but we note that there is still much room for improvement in optimizing the throughput of our routers. However, even SW ranking, our most expensive router, introduces an additional cost of no more than 0.4% when compared to GPT-4 generation, as detailed in Appendix D.

5 Conclusion

We demonstrate strong routing performance by our routers across a variety of benchmarks, spanning open-ended question answering, humanities, and math problems. By intelligently routing queries between a strong model and weak model, our routers are able to achieve significant cost savings while maintaining a high response quality.

Our results also highlight the effectiveness of dataset augmentation in improving router performance. While training routers solely on the Arena dataset results in poor performance with MMLU and GSM8K, augmenting the training data with an LLM judge or in-domain data enables our routers to outperform the random baseline across all benchmarks. The largest performance gains occur when the training data closely resembles the evaluation data, as indicated by the benchmark-dataset similarity score. We believe that this framework provides a clear and scalable path to enhancing routing performance for specific use cases.

While our work demonstrates strong results, there are a few limitations. First, although we evaluate on a diverse set of benchmarks, real-world applications may have distributions that differ substantially from these benchmarks. To this end, we show that users can collect a small amount of in-domain data to improve performance for their specific use cases via dataset augmentation. Next, while we focus on the two-model routing setting in this work, a promising future direction would be to extend this work to multiple models. Finally, in our experiments, we observe that performance between different routers trained on the same dataset can vary widely on the same benchmark without a clear explanation—we leave further investigation into this for a future work.

Acknowledgments and Disclosure of Funding

We are grateful to Kourosh Hakhamaneshi, Goku Mohandas, Arman Zharmagambetov and Anastasiia Razdaibiedina for their valuable discussions and feedback on this work. This work is in part supported by gifts from Accenture, AMD, Anyscale, Google, IBM, Intel, Microsoft, Mohamed Bin Zayed University of Artificial Intelligence, Samsung SDS, SAP, Uber, and VMware.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Meta AI. Introducing meta llama 3: The most capable openly available llm to date, 2024. Accessed: 2024-05-21.

[3] Unify AI. Unify ai, 2024. Accessed: 2024-06-30.

[4] AI@Meta. Llama 3 model card, 2024. Accessed: 2024-06-26.

[5] Anthropic. "introducing the next generation of claude", 2024. Accessed: 2024-05-22.

[6] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[8] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[9] 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.

[10] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference, 2024.

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

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[13] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations*, 2024.

[14] Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy S Liang, and Tatsunori B Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

[15] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2020.

[16] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[17] Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*, 2023.

[18] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[19] Martian. Martian router, 2024. Accessed: 2024-06-30.

[20] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[21] OpenAI. Openai pricing, 2024. Accessed: 2024-06-30.

[22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[23] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[24] Together.AI. Together.ai pricing, 2024. Accessed: 2024-06-30.

[25] Andreas Töscher, Michael Jahrer, and Robert M Bell. The bigchaos solution to the netflix grand prize. *Netflix prize documentation*, pages 1–52, 2009.

[26] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[27] 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.

[28] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[29] Weizhe Yuan, Graham Neubig, and Pengfei Liu. Bartscore: Evaluating generated text as text generation. *Advances in Neural Information Processing Systems*, 34:27263–27277, 2021.

[30] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[31] Banghua Zhu, Evan Frick, Tianhao Wu, Hanlin Zhu, and Jiantao Jiao. Starling-7b: Improving llm helpfulness & harmlessness with rlaif, November 2023.

A Arena Model Tiers

Tier	Models
Tier 0	gpt-4-0125-preview, gpt-4-1106-preview
Tier 1	gpt-4-0314, gpt-4-0613, mistral-medium, claude-1, qwen1.5-72b-chat
Tier 2	claude-2.0, mixtral-8x7b-instruct-v0.1, claude-2.1, gemini-pro-dev-api, gpt-3.5-turbo-0314, gpt-3.5-turbo-0613, gemini-pro, gpt-3.5-turbo-0125, claude-instant-1, yi-34b-chat, starling-lm-7b-alpha, wizardlm-70b, vicuna-33b, tulu-2-dpo-70b, nous-hermes-2-mixtral-8x7b-dpo, llama-2-70b-chat, openchat-3.5
Tier 3	llama2-70b-steerlm-chat, pplx-70b-online, dolphin-2.2.1-mistral-7b, gpt-3.5-turbo-1106, deepseek-llm-67b-chat, openhermes-2.5-mistral-7b, openchat-3.5-0106, wizardlm-13b, mistral-7b-instruct-v0.2, solar-10.7b-instruct-v1.0, zephyr-7b-beta, zephyr-7b-alpha, codellama-34b-instruct, mpt-30b-chat, llama-2-13b-chat, vicuna-13b, qwen1.5-7b-chat, pplx-7b-online, falcon-180b-chat, llama-2-7b-chat, guanaco-33b, qwen-14b-chat
Tier 4	stripedhyena-nous-7b, mistral-7b-instruct, vicuna-7b, qwen1.5-4b-chat, palm-2
Tier 5	koala-13b, chatglm3-6b, gpt4all-13b-snoozy
Tier 6	mpt-7b-chat, RWKV-4-Raven-14B, chatglm2-6b, alpaca-13b, oasst-pythia-12b
Tier 7	fastchat-t5-3b, chatglm-6b
Tier 8	dolly-v2-12b, stablelm-tuned-alpha-7b
Tier 9	llama-13b

B Data Contamination

We check for cross-contamination between our evaluation dataset and the preference data used for training using embedding similarity search. Embeddings are generated for the evaluation and training data using OpenAI's `text-embedding-3-small` model. For each evaluation example, we perform a similarity search across all training data with a threshold of 0.95, returning a list of contaminated examples. We discard these evaluation examples and report results on uncontaminated scores.

C Benchmark-Dataset Similarity

Let $\epsilon_B = \{b_1, b_2, \ldots, b_n\}$ be the embeddings of the prompts for a given benchmark B and $\epsilon_D = \{d_1, d_2, \ldots, d_m\}$ be the embeddings of a specific preference dataset $\mathcal{D}_{\text{pref}}$, where n and m are the total number of evaluation and preference data samples respectively. We define the *benchmark-data similarity score* $\mathcal{S}(B, \mathcal{D}_{\text{pref}})$ for each benchmark B as the average maximum similarity for each evaluation prompt across all dataset samples:

$$\mathcal{S}(B, \mathcal{D}_{\text{pref}}) = \frac{1}{n} \sum_{i=1}^{n} \max_{1 \leq j \leq m} \frac{b_i \cdot d_j}{\|b_i\| \|d_j\|} \tag{13}$$

We opt to use only the maximum similarity score because having a small number of samples of preference data that are very similar to the user's query is most valuable for efficient query routing, as opposed to having many samples that are less similar to the user prompt.

D Cost Calculation

Since our evaluations are performed with the `gpt-4-1106` endpoint, we use its pricing ($10 per 1 million input tokens and $30 per 1 million output tokens) in our analysis. For the sake of simplicity, we assume the routers will be mostly handling short prompts in a single turn setting. We find the average input prompt in the training set to be 95 tokens long, and the average output responses to be 264 tokens long. This means the input/output tokens ratio is roughly $\frac{95}{264}$. Using these information, we estimate the average cost of using GPT-4 to be: $\frac{\left(\frac{95\times10}{1,000,000}+\frac{264\times30}{1,000,000}\right)\times1,000,000}{95+264} \approx 24.7$ USD per 1 million tokens. For Mixtral 8x7B, we assume the same price for both input and output tokens, which makes the average cost $0.24 USD per 1 million tokens.

E Independent Benchmarks

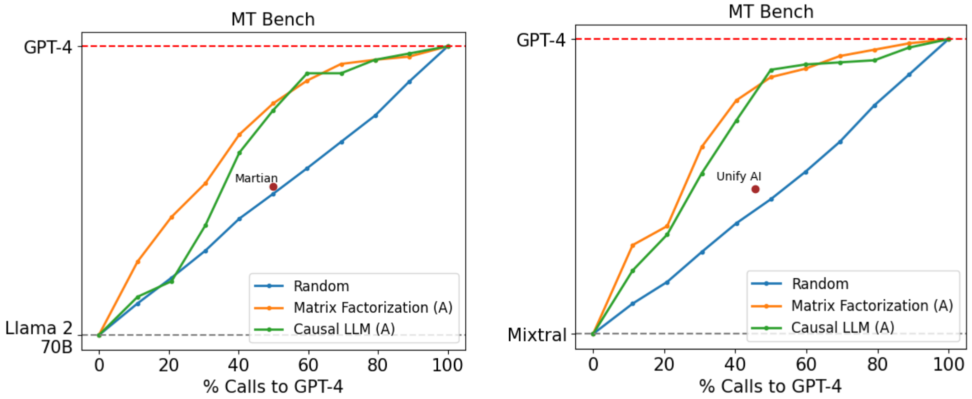

Figure 2: Performance of our routers as compared to other routing systems on MT Bench. Our routers demonstrate competitive performance, achieving stronger performance than existing routers for the same cost.

In Figure 2, we present the performance of our best-performing routers on MT Bench as compared to Unify AI [3] and Martian [19], two existing commercial offerings for LLM routing. Here, we route between `gpt-4-turbo-2024-04-09` [20] as our strong model, and `mixtral-8x7b-instruct-v0.1` [16] or `llama-2-70b-chat` [26] as our weak model depending on the available models. For Unify AI, we select the best-performing router configuration on the user dashboard and use it for benchmarking. For Martian, we optimize for performance and specify the maximum cost per million tokens as \$10.45, approximating this value using public inference costs [21, 24] based on a 1:1 input:output token ratio such that 50% of calls are routed to GPT-4. Both the matrix factorization router and causal LLM routers perform very competitively when trained on $\mathcal{D}_{\text{arena}}$ augmented with $\mathcal{D}_{\text{judge}}$, achieving the same performance as these routers with up to 40% fewer calls routed to GPT-4.

F Additional Plots

We include additional plots for all results presented in Section 4.1.

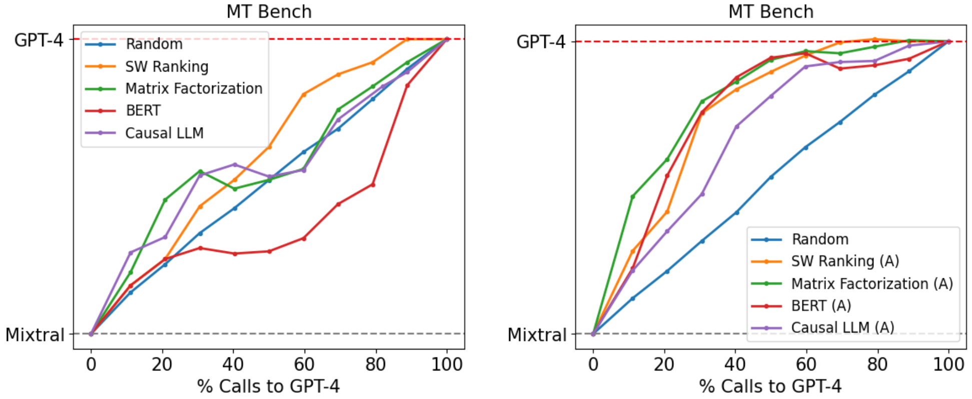

Figure 3: MT Bench performance for all routers.

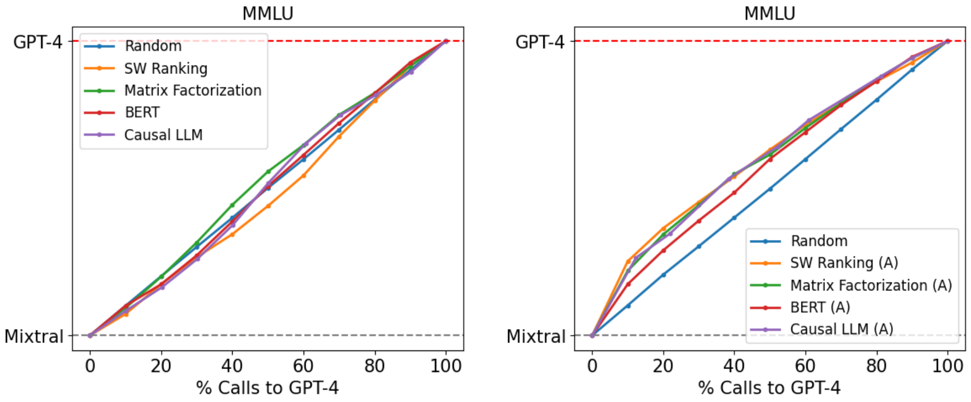

Figure 4: 5-shot MMLU performance for all routers.

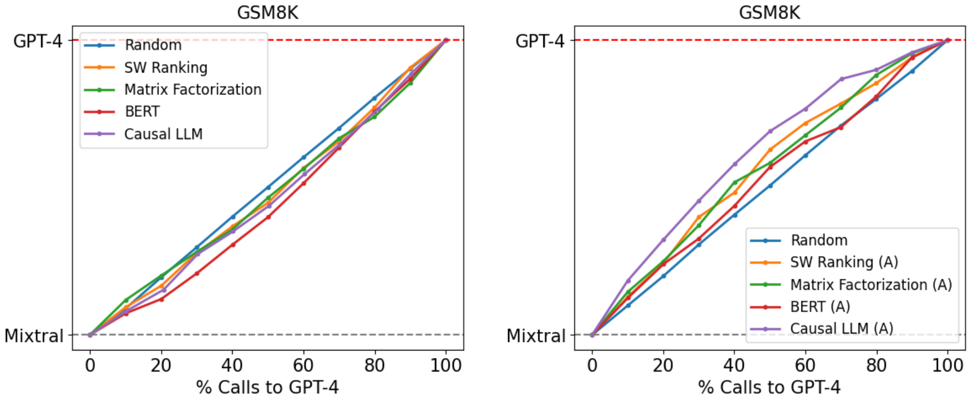

Figure 5: 8-shot GSM8K performance for all routers.

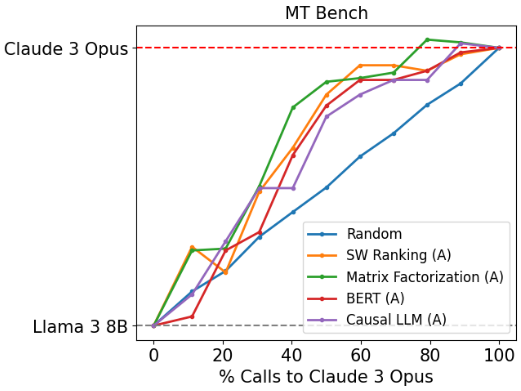

Figure 6: MT Bench performance for all routers when routed to Claude 3 Opus and Llama 3 8B instead, without any retraining.