TensorOpera Router: A Multi-Model Router for Efficient LLM Inference

Dimitris Stripelis[1], Zijian Hu[1], Jipeng Zhang[1], Zhaozhuo Xu[1], Alay Dilipbhai Shah[1], Han Jin[1], Yuhang Yao[1], Tong Zhang[1], Salman Avestimehr[1], Chaoyang He[1],
[1]TensorOpera, Inc., Palo Alto, CA, USA
Correspondence: contact@tensoropera.com

Abstract

With the rapid growth of Large Language Models (LLMs) across various domains, numerous new LLMs have emerged, each possessing domain-specific expertise. This proliferation has highlighted the need for quick, high-quality, and cost-effective LLM query response methods. Yet, no single LLM exists to efficiently balance this trilemma. Some models are powerful but extremely costly, while others are fast and inexpensive but qualitatively inferior. To address this challenge, we present *TO-Router*, a non-monolithic LLM querying system that seamlessly integrates various LLM experts into a single query interface and dynamically routes incoming queries to the most high-performant expert based on query's requirements. Through extensive experiments, we demonstrate that when compared to standalone expert models, *TO-Router* improves query efficiency by up to 40%, and leads to significant cost reductions of up to 30%, while maintaining or enhancing model performance by up to 10%.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable performance across a diverse set of challenging domain-specific tasks (Beeching et al., 2023). However, no single LLM can outperform all others across every task and use case (Shnitzer et al., 2023). Recent works (Hu et al., 2024; Ong et al., 2024; Ding et al., 2024) highlight the urgent need for efficient tools that can unify the expertise of multiple LLMs, combining them into a single cohesive unit. Such tools can allow enterprises to develop applications, e.g., for customer support, on top of a single endpoint that can integrate multiple domain experts and intelligently route any query to the most suitable expert.

However, due to the high costs and latency involved in querying LLM experts hosted at various providers (Chen et al., 2023), it is essential for such multi-LLM querying tools to efficiently and economically direct queries to the most suitable expert. This requires balancing three key factors: query throughput, monetary cost, and model performance — a challenge we refer to as the multi-LLM routing trilemma.

Our aim is to provide an empirical solution to this trilemma by showcasing the potential of a multi-LLM routing system that improves this balance. We propose an LLM routing system, called *TensorOpera-Router* (hereinafter referred to as *TO-Router*), to explore the feasibility of building a multi-LLM routing model that leverages the collective power of multiple LLM experts. *TO-Router* aims to efficiently, inexpensively, and accurately answer query prompts by selecting the most cost-effective and suitable LLM from a diverse set of expert models. Our contributions are as follows:

- We empirically demonstrate the promise of different routing methods developed through the *TO-Router* system in balancing query execution time, query cost, and model performance, leading to significant gains.
- We show that, on average, our routing system outperforms standalone model experts.
- We demonstrate that routing methods trained to learn the embedding query space outperform naive routing methods.
- We present a routing method based on a pretrained BERT model that exhibits the best performance.

2 Background & Related Work

Model Routing. Depending on the mechanism used by routing methods to decide the most suitable LLM(s) to answer a given prompt, two distinct routing categories have been recently introduced: *predictive/classification routers*, which do not generate LLM outputs in advance, but instead, they

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track, pages 452–462
November 12-16, 2024

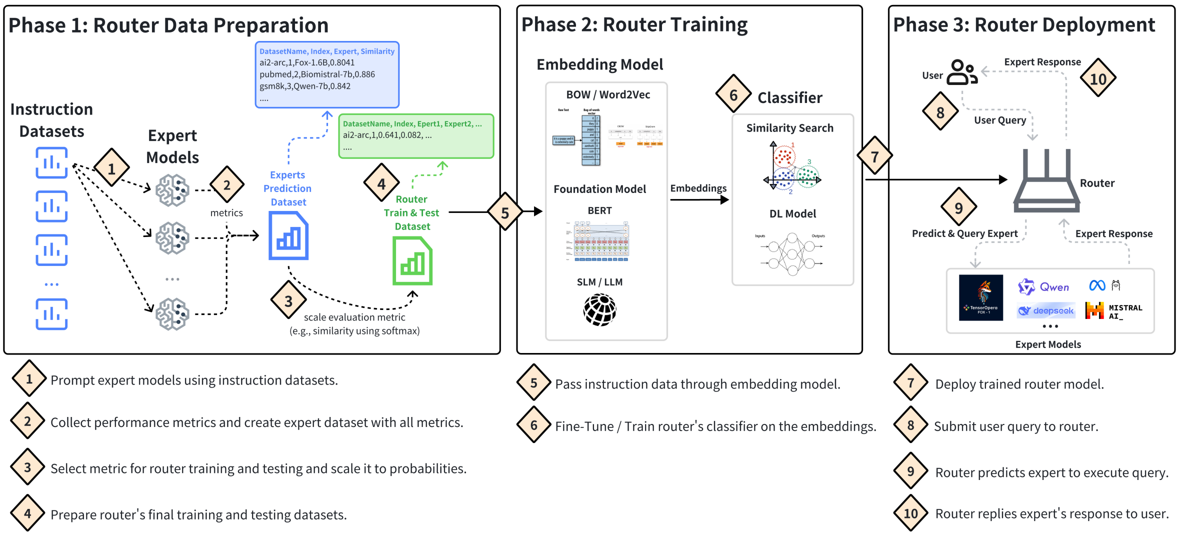


Figure 1: TO-Router system's overview of router data preparation, router model training and deployment pipelines.

predict the best LLM to handle a given prompt based on specific performance metrics (Hu et al., 2024; Ong et al., 2024; Srivatsa et al., 2024) and *cascading routers*, which refer to routing methods that process a query request by executing it over a series or combinations of LLMs (Chen et al., 2023) until specific quality criteria are met. To train the predictive routers, different training methods have been recently introduced that leverage data augmentation techniques and human preference data (Ong et al., 2024) or existing benchmark datasets (Shnitzer et al., 2023) to improve routing predictions. In this work, we too develop and evaluate predictive routing methods trained on standardized benchmark datasets to efficiently classify and direct query prompts to the best LLM expert.

Mixture-of-Experts. A typical MoE architecture (Jordan and Jacobs, 1994) consists of a set of expert models trained to specialize in different data regions and a gating network model that determines the contribution of each expert to the final prediction. Recently, MoEs have witnessed wide adoption in the LLM domain as well, where multiple MLP experts are integrated into encoder and decoder blocks, to boost the training of extremely large networks (Shazeer et al., 2017; Jiang et al., 2024; Fedus et al., 2022). Similar to these MoE approaches, the LLM routing methods can be seen a special case of an MoE architecture, where the predictive routing model is the gating mechanism and the pool of LLMs the set of available experts. However, unlike MoE architectures that route to homogeneous experts, our approach routes to heterogeneous domain experts of varying sizes.

Ensemble Learning. Model routing also bears similarities with ensemble machine learning (Zhou, 2012) techniques that seek to provide better predictive outcomes by combining the predictions of multiple models. A key distinction between routing and ensemble techniques, like bagging (Breiman, 1996), and boosting (Freund and Schapire, 1997), is that models participating in an ensemble are typically trained on the (whole or subsets of) same dataset and therefore assumed to have a similar expertise. However, the router predicts and retrieves the predictions out of a varying set of LLMs experts that have been trained on highly diverse sets of data distributions.

3 TensorOpera Router System Overview

To effectively learn and deploy a multi-LLM routing model, a sequence of different critical development phases need to be executed, from data preparation to router model training and evaluation and model deployment/serving. The proposed TO-Router system's end-to-end pipeline shown in Figure 1 facilitates the development of these phases and in practice has helped to swiftly develop, prototype and deploy different model routing methods into real-world settings. [1]

Phase 1: Router Data Preparation. The generation of the training and testing dataset for the routing model is a multi-step process. First, we need to find the appropriate domain specific (e.g.,

[1] We plan to release the source code as open-source soon.

bio, coding, physical sciences) instruction datasets and model experts to which we want the routing model to learn propagating relevant query prompts. Thereafter, we perform a forward pass over each expert model (step 1) to collect the associated metrics required to train and test the performance of the routing model and create the experts prediction dataset (step 2), i.e., for every prompt in each dataset we query each expert individually. In this work, we collect the following metrics per instruction prompt: {negative log likelihood, BERT similarity score (BERTSim), inference time in seconds, total input tokens, total output tokens}; for more details on these metrics, please see section 4.4. Once the expert prediction dataset is created, we select one of the collected metrics to generate soft labels (step 3) and prepare the final training and testing dataset for the routing model (step 4). In the current work, we use the BERTSim scores to create soft labels and train the routing expert model classifier. We use soft labels, since we want the routing model to learn the ranking of the experts in terms of their prediction performance. To generate the soft labels of each expert model and for each instruction record, we pass the selected metric (e.g., similarity score, log loss), through a softmax function with temperature. For instance, for the r-instruction record, the expert (class) softmax probability ϕ_r is given by: $\phi_r(\mathbf{x};T) = \frac{\exp(\frac{x_i}{T})}{\sum_{j=1}^{E}\exp\left(\frac{x_j}{T}\right)}$, where E is the total number of experts, T is the temperature value, and $\mathbf{x} = (x_1, x_2, \ldots, x_E)$ is the vector of metric scores. In our evaluation, we generate expert's soft labels based on the BERT similarity scores and with a temperature value of $T = 10$.

Phase 2: Router Training. Once the router's training and testing dataset is created, we pass the instruction records through the router's embedding model, e.g., Bag-of-Words, TF-IDF, BERT or other small or large language models, to create their vectorized representation (step 5). Then, we use the generated embeddings to train the prompt-to-expert classifier (step 6), using non-parametric, supervised learning approaches (e.g., kNN), classical deep learning models (e.g., MLP) or more advanced language sequencing pre-trained models (e.g., BERT). Even though our approach for creating the soft labels is versatile and can be applied to any metric or combination of metrics, in this work, we only consider the BERTSim score as part of the MLP and BERT routers' training cost function, since all expert models are deployed on the same hardware, and therefore throughput and cost per token are similar across all experts. More information on these routing models is provided in section 4.3. It is important to note that, during training and testing, we only consider the best, most suitable (top-1) expert, but as it is also discussed in Section 5, our approach can also be extended to combine the responses of multiple (top-k) experts.

Phase 3: Router Deployment. When the final routing model is trained, the model is deployed as a standalone endpoint on the platform (step 7), ready to receive user queries (either through CLI or web interface). Whenever a new user query is submitted, the router first tokenizes and encodes the text of the incoming query prompt using the tuned embedding model from Phase 2 (step 8). Subsequently, the router performs a forward pass over the trained/fine-tuned classification model (e.g., MLP, BERT) and predicts the most relevant expert model (step 9). Depending on which expert model the classification model predicts, the router selects the respective expert-prompt adaptor to submit and execute the query. Once query execution completes, the router receives the reply from the expert model and forwards it back to the end user (step 10). Throughout the router's deployment time, the platform provides the necessary monitoring capabilities to troubleshoot and tune the routing model, such as number of requests, queries' semantic context, expert models hitting frequency, and total costs.

4 Experiments

In this section we discuss the expert models, benchmark datasets, routing methods and metrics we considered to evaluate the TO-Router system.

4.1 Expert Models

We choose several representative models across different domains as the expert models to verify the effectiveness of our routing method in the TO-Router system. For the Biomedical domain, we selected two variants from Llama-3-8B (**BioLlama-7B**) (Shao et al., 2024) and Mistral-7B (**BioMistral-7B**) (Labrak et al., 2024) models [2]. Both models achieve excellent performance across many biomedical evaluation benchmarks. In the code domain, we select Meta's officially released Llama2-7B (**CodeLlama-7B**) (Roziere et al., 2023) variant trained on code datasets. In the general instruction-following domain, we incorporate three

[2]We refer to each model using its name in bold fonts.

instruction-tuned versions of LLMs across different sizes, i.e., **Fox-1.6B** (TOAI, 2024) a recently introduced powerful small language model, Mistral-7B-Instruct (**MistralAI-7B**) (Jiang et al., 2023), and Qwen-7B-Instruct (**Qwen-7B**) (Yang et al., 2024). Finally, for the math domain, we choose a strong reasoning model trained on large amounts of math documents, MathDeepSeek-7B-Instruct (**MathDeepSeek-7B**) (Guo et al., 2024). More details regarding models' architecture and fine-tuning please please see section D in the Appendix.

4.2 Datasets

All the datasets listed here are widely used by LLM developers (Touvron et al., 2023a,b; Jiang et al., 2023) to evaluate model performance in commonsense reasoning, coding, and medical domains. To generate the final training and testing data for the investigating routing methods, we gather all records together from all datasets and perform a stratified 80% train, 20% test split per dataset.

Ai2-ARC (Clark et al., 2018). The Ai2-ARC dataset consists of 7,787 natural science questions designed for standardized tests. We use its challenge partition with 2,590 samples, which includes only those questions that were answered incorrectly by both a retrieval-based algorithm and a word co-occurrence algorithm.

GSM8k (Cobbe et al., 2021). GSM8k is a high-quality dataset of grade school-level math word problems, covering relatively simple math concepts with 7,473 training and 1,319 testing samples.

MBPP (Austin et al., 2021). The MBPP dataset contains 974 basic programming problems suitable for entry-level programmers. It also includes text descriptions of the problems and test cases for functional correctness verification.

PubMedQA (Jin et al., 2019). The PubMedQA dataset is a biomedical question-answering dataset designed for answering research questions with yes/no/maybe responses. It contains 1,000 manually labeled question-answer pairs for cross-validation and testing.

4.3 Routing Methods

Below, we describe the various predictive and non-predictive routing methods we consider in our evaluation. [3]

[3] To ensure routing models are cost-effective and economically viable, we omit LLM-based routers from our current evaluation setting.

Zero-Router. Following the work of (Hu et al., 2024), we also evaluate the performance of the routing methods against the average performance of the available LLMs without any routing logic (lower bound), i.e., no-routing approach.

Optimal. We compare against two optimal cases (upper bounds), one refers to the optimal BERTSim performance per dataset (shown in Figure 2a), and the other to the optimal performance recorded across all three evaluating dimensions (i.e., cost, throughput, model performance, shown in Figure 3). In the former case, the optimal value is measured by averaging the best BERTSim score recorded for every test query by any expert. In the latter case, the optimal set of values is the minimum cost, maximum throughput and maximum performance recorded by any expert model or router method.

Random-Router. To evaluate the performance of a random router, for every test query we randomly pick an expert to execute the query. After performing this step for all test queries, we repeat the entire process for 10 times. Let $\mathbf{E} = (e_1, e_2, \ldots, e_N)$ be the collection of all experts, we randomly select an expert from $\mathbf{E}$ in each trial. Let e_i^j denote the i expert randomly selected in the j-th trial, then the entire random expect selection process can be represented as: $\{e_i^1, \ldots, e_i^{10}\}$. Once the collection of random experts is assembled, we submit the test query to each expert and collect all measurements to compute the evaluation metrics.

kNN-Router. The kNN-Router first encodes all training queries $\mathbf{q_i} \in D^t$ using a sentence transformer. Then, for every test query, $\mathbf{q_t}$, it finds its closest training query $\mathbf{q_i'}$ in terms of cosine similarity in the embedding space and subsequently executes the test query using the expert that exhibited the best performance for the most relevant training query. The best performing expert $\mathbf{e_i'}$ is the expert whose BERTSim score is the highest out of all the training query's experts, $q_i'(E)$:

$$\mathbf{q_i'} = \min_{i \in D^t}\left(\frac{\mathbf{q_i} \cdot \mathbf{q_t}}{\|\mathbf{q_i}\|\|\mathbf{q_t}\|}\right)$$

$$\mathbf{e_i'} = \max_{j \in q_i'(E)}(BERTSim_j)$$

A schematic flow of the 1NN-Router's embedding similarity and expert selection is also shown in Figure 5. Given that we only need to find the most similar training query to a given test query, we subsequently refer to this method as *1NN-Router*.

MLP-Router. To learn our predictive MLP-Router, we use a simple 2-layer perceptron:

$$y_k = \phi\left(\sum_{j=1}^{m} w_{jk}^{(2)} \sigma\left(\sum_{i=1}^{n} w_{ij}^{(1)} x_i + b_j^{(1)}\right) + b_k^{(2)}\right)$$

To train the MLP model, we convert the training queries into their vector representation by fitting a Bag-of-Words model. To learn the ranking of experts in terms of prediction performance, we use cross entropy loss on the scaled BERTSim scores. We used ReLU (σ) and softmax (ϕ) as the hidden and output layers' activation function, respectively.

BERT-Router. To learn the BERT-Router, we performed a full parameter fine-tuning on a BERT model (approx. 110M parameters) for sequence classification. We appended a classification head with a softmax activation funciton on top of BERT's final hidden layer outputs to map the BERT embeddings H to the number of experts (classes):

$$y = \text{softmax}(WH + b), \; H = \text{BERT}(X)$$

To fine-tune BERT, we first tokenize and encode all input training queries' text sequences X using the BERT tokenizer and then update the pre-trained BERT model weights for a small number of epochs using cross entropy loss. Similar to the MLP-Router model, we train BERT-Router using the soft labels created by the scaled BERTSim scores.

4.4 Evaluation Criteria

All expert models and routing methods are evaluated on four dimensions: (1) total inference cost, (2) throughput, (3) BERT similarity score, and (4) negative log loss (NLL).

Total Inference Cost. For any expert model the total cost to execute a given test query is measured based on the input and output token costs. For a model m that was prompted with a sequence of test queries that were used a total number of T_i input tokens, and the model generated a total number of T_o output tokens, with a c_i and c_o cost per 1 million input and output tokens, respectively, the total cost for the entire test query sequence is measured by: $C_m = \frac{T_i}{1e6} * c_i + \frac{T_o}{1e6} * c_o$. In the case of the routing methods that did not use one single model to answer the sequence of testing queries but routed different testing queries to different expert models M, the total cost is measured as: $C_r = \sum_{m \in M} C_m$. To measure the querying of standalone deployed expert models, we handpicked the price per million input and output tokens from different model providers. Table 1 shows the cost of input and output token per model architecture.

Model Type	$$ / 1M Input Tokens	$$ / 1M Output Tokens
DeepSeek-8B	$0.14	$0.28
Fox-1.6B	$0.20	$0.20
Llama-8B	$0.20	$0.20
Mistral-8B	$0.25	$0.25
Qwen-7B	$0.20	$0.20

Table 1: Price per million input and output tokens for different model architectures.

Throughput. To measure the querying execution performance of a expert model and of different routing methods for the entire test query set, we compute the throughput for each query as the fraction of total output tokens T_m^o, generated by each model m, over the inference time in seconds, i.e., time from query submission to query completion, t_m^s. Specifically, the throughout for a single test query i is measured as $\tau_i = \frac{T_m^o}{t_m^s}$. For the entire set of test queries N, the mean throughput $\tilde{\tau}$ is computed as: $\tilde{\tau} = \frac{1}{N} \sum_i^N \tau_i$.

BERTSim. Given that each expert model uses its own vocabulary and tokenizer and to ensure that there is an equitable comparison between the responses generated by each expert, we evaluate the vectorized text similarity between the ground truth and the predicted answer of an expert through the cosine distance on the BERT embeddings; during computation the expert response is used as is without any post-processing. Such a vector representation allows for a soft measure of similarity (Zhang et al., 2019). We refer to this similarity score as BERTSim (Zhang et al., 2019). The cosine similarity of a reference (ground truth) vector x_i and a candidate (predicted) vector $\hat{x}_j$ is computed as: $\frac{\mathbf{x}_i^\top \hat{\mathbf{x}}_j}{\|\mathbf{x}_i\| \|\hat{\mathbf{x}}_j\|}$. For every expert model and routing method we measure the BERTSim score across all test queries and we compute the final BERTSim score as the mean of all scores.

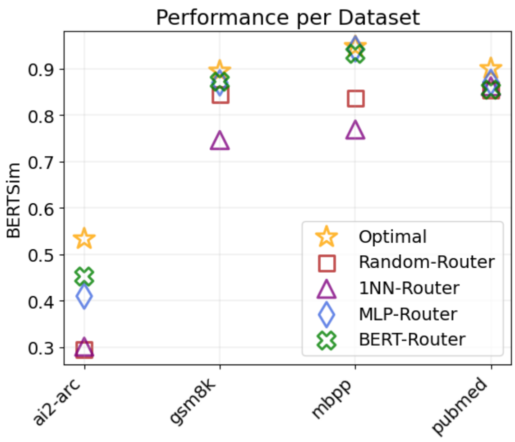
Negative Log-Likelihood. We use the Negative Log-Likelihood (NLL) to measure the quality of the probabilistic predictions made by each expert model. Lower NLL values are indication that the model is assigning higher probabilities to the true classes and therefore reflecting better performance. In principle, a single sequence's NLL is defined as:

$$\mathcal{L}_{\text{NLL}} = -\sum_{t=1}^{T} \log P(y_t \mid X, y_{1:t-1})$$

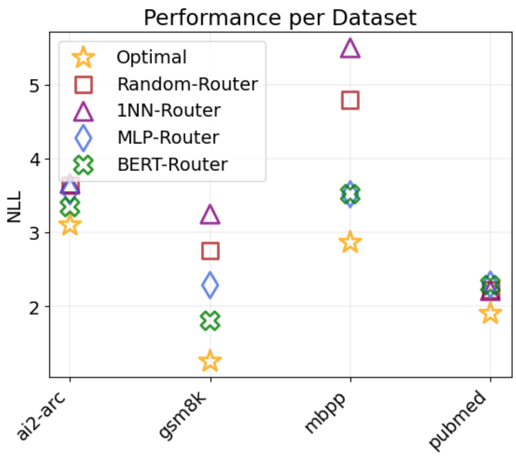
where $P(y_t \mid X, y_{1:t-1})$ is the predicted probability of the t-th token in the sequence given the input sequence X and the previous tokens $y_{1:t-1}$. In our evaluation, we measure the mean NLL over the generated sequence of every expert model and routing method across all test queries.

4.5 Evaluation

To systematically evaluate all investigating expert models in terms of query response times, we deployed each model on a machine employed with 8 NVIDIA DGX H100 GPUs. [4] Figures 2a and 2b show the BERTSim score and NLL value comparison between all routing and optimal methods.



(a) BERT Similarity Score.



(b) Negative Log-Likelihood.

Figure 2: Router performance per dataset.

From the router vs. router comparison in Figures 2a and 2b, it is shown that naive methods, such as Random-Router or 1NN-Router that do not learn

Model / Router	Total Cost	Throughput	BERTSim	NLL
BioLlama-8B	\$0.195	155.613	0.686	3.408
BioMistral-8B	**\$0.125**	208.399	0.669	3.581
CodeLlama-7B	\$0.156	102.993	0.694	3.299
Fox-1.6B	**\$0.118**	**214.925**	**0.761**	**2.958**
MathDeepSeek-7B	\$0.138	187.166	0.746	3.286
MistralAI-7B	\$0.223	89.587	0.694	4.205
Qwen-7B	\$0.164	114.008	0.698	**2.326**
Random-Router	\$0.143	**209.171**	0.715	3.316
1NN-Router	\$0.131	208.399	0.669	3.581
MLP-Router	\$0.147	177.508	**0.773**	3.164
BERT-Router	**\$0.122**	**213.145**	**0.783**	**3.091**

Table 2: Total querying cost, mean throughput and cosine similarity between predicted and expected answers per model and router considering all the four benchmark datasets. Box coloring represents the following ranking column-wise: rank 1, rank 2, rank 3.

the embedding space can lead to suboptimal performance, cf. 0.3 BERTSim score for Random- and 1NN- Routers to 0.4 and 0.45 of MLP- and BERT-Routers in the Ai2-ARC dataset. Analogously, when it comes to train routing models that learn the embedding space, cf. BERT-Router to MLP-Router, more complex routing methods (i.e., BERT-Router) can lead to better outcomes and match closer the optimal performance, especially in challenging domains like GSM8K, cf. BERT-Router's NLL value of 1.803 to MLP-Router's 2.286.

To conduct a more thorough evaluation between expert models and routing methods, in Table 2, we record all the numerical values collected throughout our experiments in terms of total monetary cost, query throughput, BERTSim score and NLL value. For every evaluating dimension, we also highlight with different colors the top-3 positions/rankings. The recorded values for the Zero-Router and the Optimal across all four dimensions are, Zero-Router: {\$0.161, 153.242, 0.707, 3.295} and Optimal: {\$0.118, 214.925, 0.783, 2.326}; we do not report these values in the table to emphasize the ranking between routing methods and standalone models. The MistralAI-7B exhibits the worst performance across all expert models, while the more recent small language model, Fox-1.6B, has the best performance across all expert models and evaluating dimensions.

By using as a reference routing method the BERT-Router approach and baseline the mean performance of all standalone model experts (i.e., the Zero-Router), we find that the BERT-Router leads to a close of 30% cost reduction and 40% query inference throughput increase compared to no routing at all. At the same time though, BERT-Router

[4] Due to production demands, we could reserve only 1 GPU to perform the evaluation. Hence, we resorted to evaluate models with 7B params hosted on a single GPU, since larger models (e.g., 70B params) would require at least 2 GPUs.

is capable of maintaining or slightly enhancing the average mean model performance, by a 11% in terms of BERT similarity score and lead to a 6% NLL reduction.

To further analyze the optimization trilemma problem w.r.t. total monetary cost (x-axis), query throughput (y-axis) and model performance (z-axis), Figure 3 provides a 3D visualization of the tree different metrics. As it is clearly shown in the Figure, the BERT-Router method outperforms all other expert models and routing methods across all three evaluation criteria, while almost matching the optimal performance.

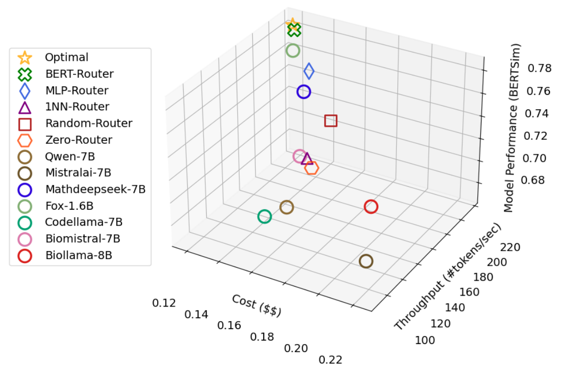


Figure 3: A holistic view of model performance, throughput and total querying cost for standalone deployed expert models and different routing methods.

Independent of Fox's and other expert models' performance, the collective model power provided by the routing methods, especially of the BERT-Router method, outperforms any other standalone expert model. This can also be seen in the query per expert assignment heatmap shown in Figure 4, where we record the number of test queries answered by each expert model for every routing method. From the reported values, it is apparent that both the MLP-Router and the BERT-Router route most of the test queries to the Fox-1.6B small language model, which is similar to the behavior observed by the Optimal (oracle) approach. However, other routing approaches like the Random-Router and 1NN-Router, distribute almost equally the number of queries across all model experts.

Overall, our evaluation shows that routers can match or outperform standalone large language model experts (e.g., BERT-Router vs. MistralAI-7B, BioLlama-8B). The BERT-router model is highly efficient, with just 110M parameters — 15 times smaller than the Fox SLM and 70 times smaller than the studied LLMs — making it ideal for production. While we didn't assess routers' performance against extremely large models, our results suggest that our routing and evaluation methods are applicable to larger models and are not tied to the ones studied in this work.

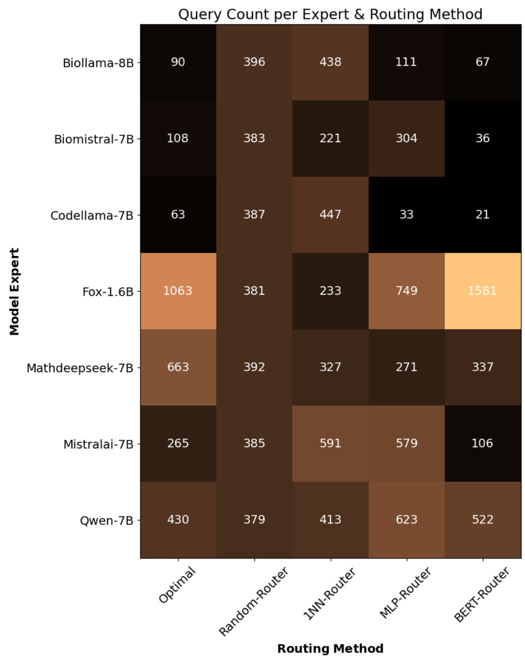


Figure 4: Number of test queries allocated to each model expert by each routing method.

5 Conclusion

We presented for the first time our multi-LLM routing system, called *TO-Router*. Through the TO-Router system, users can easily interact with multiple LLM expert models hosted at the same or across multiple platform providers, without having to restrict themselves to a single monolithic LLM system. At the same time, users can overall benefit from significant cost savings (up to 30%) and improved query response times (up to 40%) while maintaining or enriching (up to 10%) model performance. As part of our immediate future plan we aim to evaluate the feasibility of dynamically adding and removing model experts during router's endpoint deployment, and test the routing efficacy of small and large language pre-trained models. Finally, we also plan to evaluate approaches where we combine the responses of top-k experts into one instead of returning the response of a single expert.

References

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

Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. 2023. Open llm leaderboard. *Hugging Face*.

L Breiman. 1996. Bagging predictors machine learning 24 (2), 123-140 (1996) 10.1023. *A: 1018054314350*.

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

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

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

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. 2024. Hybrid llm: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618*.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. 2024. Routerbench: A benchmark for multi-llm routing system. *arXiv preprint arXiv:2403.12031*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

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. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W Cohen, and Xinghua Lu. 2019. Pubmedqa: A dataset for biomedical research question answering. *arXiv preprint arXiv:1909.06146*.

Michael I Jordan and Robert A Jacobs. 1994. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214.

Yanis Labrak, Adrien Bazoge, Emmanuel Morin, Pierre-Antoine Gourraud, Mickael Rouvier, and Richard Dufour. 2024. Biomistral: A collection of open-source pretrained large language models for medical domains. *arXiv preprint arXiv:2402.10373*.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. 2024. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Yu Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

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

KV Srivatsa, Kaushal Kumar Maurya, and Ekaterina Kochmar. 2024. Harnessing the power of multiple minds: Lessons learned from llm routing. *arXiv preprint arXiv:2405.00467*.

TOAI. 2024. tensoropera/Fox-1-1.6B · Hugging Face — huggingface.co. https://huggingface.co/tensoropera/Fox-1-1.6B. [Accessed 14-10-2024].

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Zhi-Hua Zhou. 2012. *Ensemble methods: foundations and algorithms*. CRC press.

A kNN-Router Diagram

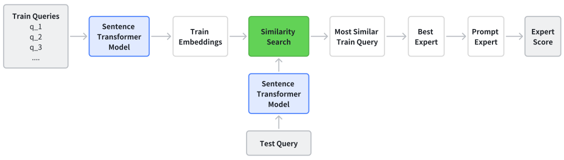


Figure 5: A flow diagram of the embedding similarity approach used by the 1NN-Router.

B Router Models Data Preparation

To generate experts' soft labels to train the MLP and BERT-Router models, we used the BERT similarity scores and set the temperature value of the softmax function to 10, i.e., $T = 10$. To compute the closest training query to a given test query in the case of the 1NN-Router, we compute the queries' embeddings using the sentence transformer library. [5]

C Router Models Training Hyperparameters

The total number of experts is 7. The MLP-Router's hidden layer size is 256. The random seed for all experiments is set to 42. The applied optimizer for training both the MLP and BERT routers is Adam with weight decay, the learning rate is set to $5e-3$ and $5e-5$, respectively. We also applied L2 norm regularization with $\lambda = 1e-4$. The batch size is set to 8 and the total number of training (MLP model) and fine-tuning (BERT model) is set to 5 epochs. The BERT model for the router is *bert-base-uncased*. To counter dataset class/expert imbalance we observed while generating the training and testing datasets, i.e., an expert model might be more suitable to answer many more queries than other experts, we used a sample weighting function, with the weight of each sample being the inverse proportion count of samples per class in the entire training dataset, i.e., the total weight sample proportion for each class/expert i across all experts E, is measured as $w_i = \frac{\sum_{j \in E} |D_j|}{|D_i|}, \forall i \in E$, with the final weight value per training sample being equal to $w_i = \frac{w_i}{\sum_{j \in E} |w_j|} \forall i \in E$.

D Expert Model Resources

Below, we provide details regarding the internal architecture and type of models we used as our expert models in this study. For every instructed model, if not otherwise specified, we set the maximum tokens generation length to 512, the temperature to 0.7, and the top-p parameter to 0.95.

- **BioLlama-7B** [6]:This model is an advanced Llama-3-based model designed specifically for the biomedical domain. With policy optimization and a custom medical instruction dataset, it outperforms even the ChatGPT API. Following the recommended parameters, we set max new tokens to 256, temperature to 0.1 and top-p to 0.9.

- **BioMistral-7B** [7]: This Mistral-based model, pre-trained using textual data from PubMed Central Open Access, is well-suited for medical domains and achieves performance comparable to the ChatGPT API across all medical evaluation benchmarks.

- **CodeLlama-7B** [8]: This model adapts the Llama-2-7B model with a large collection of code datasets, incorporating an infilling training objective and long input context subsets.

- **Fox-1.6B** [9]: Fox-1 is a decoder-only transformer-based small language model with 1.6B parameters, developed by TensorOpera AI. Fox-1-Instruct-v0.1 is an instruction-tuned version with an 8K native context length, finetuned with 5B tokens of instruction-following and multi-turn conversation data.

- **Mistral-7B-Instruct** [10]: This model is an officially released instruct fine-tuned version of the Mistral-7B-v0.2.

- **Qwen-7B-Instruct** [11]: This model is an officially released instruct fine-tuned version of the Qwen2-7B.

[5] https://www.sbert.net/docs/quickstart.html

[6] https://huggingface.co/aaditya/Llama3-OpenBioLLM-8B

[7] https://huggingface.co/BioMistral/BioMistral-7B

[8] https://huggingface.co/codellama/CodeLlama-7b-hf

[9] https://huggingface.co/tensoropera/Fox-1-1.6B-Instruct-v0.1

[10] https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

[11] https://huggingface.co/Qwen/Qwen2-7B-Instruct

- **MathDeepSeek-7B** [12]: This model, initialized with DeepSeek-Coder-v1.5 7B, continues pre-training on math-related tokens sourced from the web, achieving impressive scores on the competition-level MATH benchmark. Following the recommended parameters, we set max new tokens to 512, top-k to 50 and top-p to 0.95.

[12] https://huggingface.co/deepseek-ai/deepseek-math-7b-instruct