UNIVERSAL MODEL ROUTING FOR EFFICIENT LLM INFERENCE

Wittawat Jitkrittum
wittawatj@gmail.com

Harikrishna Narasimhan
hnarasimhan@google.com

Ankit Singh Rawat
ankitsrawat@google.com

Jeevesh Juneja
jeeveshjuneja@google.com

Congchao Wang
ccwang@vt.edu

Zifeng Wang
zifengw@google.com

Alec Go
ago@google.com

Chen-Yu Lee
chenyulee@google.com

Pradeep Shenoy
shenoypradeep@google.com

Rina Panigrahy
rinap@google.com

Aditya Krishna Menon
adityakmenon@google.com

Sanjiv Kumar
sanjivk@google.com

Google

ABSTRACT

Model routing is a simple technique for reducing the inference cost of large language models (LLMs), wherein one maintains a pool of candidate LLMs, and learns to route each prompt to the smallest feasible LLM. Existing works focus on learning a router for a *fixed* pool of LLMs. In this paper, we consider the problem of *dynamic* routing, where *new, previously unobserved* LLMs are available at test time. We propose `UniRoute`, a new approach to this problem that relies on representing each LLM as a *feature vector*, derived based on predictions on a set of representative prompts. Based on this, we detail two effective instantiations of `UniRoute`, relying on *cluster-based* routing and a *learned cluster map* respectively. We show that these are estimates of a theoretically optimal routing rule, and quantify their errors via an excess risk bound. Experiments on a range of public benchmarks show the effectiveness of `UniRoute` in routing amongst 30+ unseen LLMs.

1 INTRODUCTION

Large language models (LLMs) have seen a flurry of development (Radford et al., 2018; 2019; Brown et al., 2020; Touvron et al., 2023; Anil et al., 2023; Grattafiori et al., 2024; DeepSeek-AI et al., 2024). While LLMs have demonstrated impressive abilities, their inference cost can be daunting (Li et al., 2024a; Wan et al., 2024; Zhou et al., 2024b). Several techniques aim to ease this issue, such as speculative decoding (Stern et al., 2018; Chen et al., 2023a; Leviathan et al., 2023), early-exiting (Schuster et al., 2022), quantisation (Chee et al., 2023), compression (Frantar & Alistarh, 2023; Agarwal et al., 2024; Rawat et al., 2024), and others (Pope et al., 2023; S et al., 2024).

Our interest is in *model routing* for efficient inference. Here, one maintains a pool of candidate LLMs of various sizes and capabilities. Given a prompt, one learns to predict the lowest-cost LLM which can reasonably address the prompt. In doing so, one can learn to use high-cost LLMs sparingly, only on the (relatively) few "hard" inputs. This is a conceptually simple but effective technique, which has seen a surge of recent interest (Hendy et al., 2023; Hari & Thomson, 2023; Ding et al., 2024; Šakota et al., 2024; Chen et al., 2024b; Hu et al., 2024b; Shnitzer et al., 2023; Wang et al., 2023; Stripelis et al., 2024; Ong et al., 2025; Zhuang et al., 2024; Srivatsa et al., 2024; Feng et al., 2024; Lu et al., 2024; Zhao et al., 2024b; Dann et al., 2024; Aggarwal et al., 2024; Lee et al., 2024a; Mohammadshahi et al., 2024; Chuang et al., 2025; Huang et al., 2025).

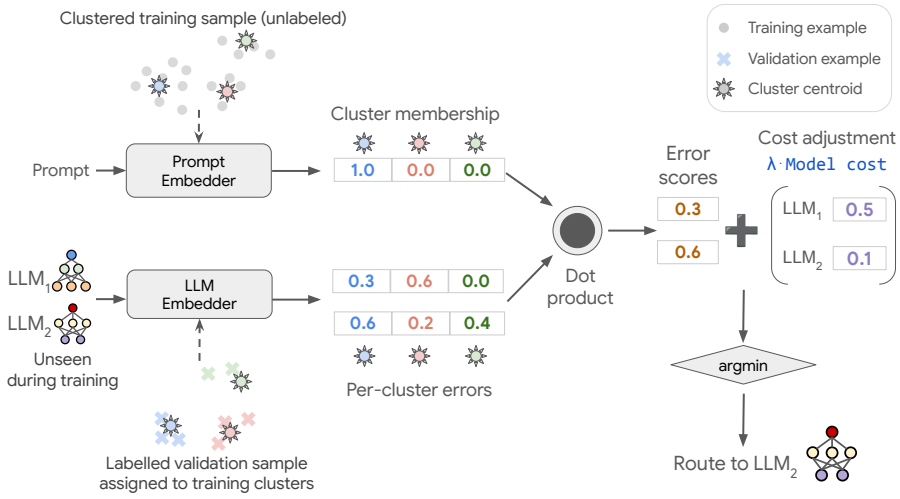


Figure 1: Illustration of our proposed `UniRoute` with a cluster-based router (see §5.1). We first perform K-means on a training set to find K centroids, and then partition the validation set into K representative clusters. Each test-time LLM can then be represented as a K-dimensional feature vector of per-cluster errors. This yields an intuitive routing rule: for each test prompt, we route to the LLM with the smallest cost-adjusted average error on the cluster the prompt belongs to. The prompt embedder may either be completely *unsupervised* (as shown in the figure), or fitted via *supervised* learning using labels from a set of training LLMs different from those seen during test time (§5.2).

Existing works largely focus on routing over a *fixed* pool of LLMs. In practice, however, the pool of LLMs can constantly change; e.g., older LLMs may be deprecated in favor of newer, more performant LLMs. To leverage such new LLMs, perhaps the simplest approach is to retrain the router. However, with frequent changes to the LLM pool, this may be impractical owing to a non-trivial overhead from repeated retraining and redeployment (see §3 for further discussion and considerations).

In this paper, we propose `UniRoute`, a new approach to this problem based on representing each LLM as a *feature vector*, derived from their *prediction errors* on a set of representative prompts. By learning a router over these LLM features, we enable generalisation to previously unseen LLMs *without* any retraining. Building on the observation that K-NN routing (Hu et al., 2024b) is a special case of `UniRoute`, we detail two concrete instantiations of `UniRoute` relying on *unsupervised* and *supervised* prompt clustering respectively. While conceptually simple, empirically, these enable effective routing with a dynamic LLM pool across several benchmarks. In sum, our contributions are:

(i) we formalise the problem setting of model routing with a *dynamic* pool of LLMs (§3);

(ii) we propose `UniRoute` (§4), a novel routing approach relying on an LLM feature representation based on its *prediction error vector* on a small set of representative prompts (§4.1);

(iii) we propose two simple yet effective instantiations of `UniRoute` based on unsupervised or supervised clustering (§5.1, §5.2, Figure 1), with an accompanying excess risk bound (§5.3);

(iv) we empirically illustrate (§6) `UniRoute` effectively handles 30+ new LLMs across multiple benchmarks (Zhuang et al., 2024; Hu et al., 2024b; Ong et al., 2025; Somerstep et al., 2025).

2 Background: Model Routing with a Static LLM Pool

Language models (LMs). Given a finite, non-empty vocabulary of *tokens* $\mathcal{V}$, a *language model* (*LM*) is a probability distribution p over sequences of tokens $\mathcal{V}^* \doteq \bigcup_{n=0}^{\infty} \mathcal{V}^n$. Our interest is in using *large* language models (*LLMs*) for prediction problems mapping *input prompts* $\boldsymbol{x} \in \mathcal{X} \subset \mathcal{V}^*$ to *targets* $\boldsymbol{y} \in \mathcal{Y}$. Our goal is to construct a *predictor* $h \colon \mathcal{X} \to \mathcal{Y}$ minimising $R(h) \doteq \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathbb{P}}\left[\ell(\boldsymbol{x}, \boldsymbol{y}, h(\boldsymbol{x}))\right]$. Here, $\ell(\boldsymbol{x}, \boldsymbol{y}, h(\boldsymbol{x}))$ denote a *loss function* measuring the *loss* of a *predicted* response $h(\boldsymbol{x})$ on a given (prompt, target) pair $(\boldsymbol{x}, \boldsymbol{y})$. For example, $\ell(\boldsymbol{x}, \boldsymbol{y}, h(\boldsymbol{x})) = \mathbf{1}(\boldsymbol{y} \neq h(\boldsymbol{x}))$ penalises responses that are not exact string matches of the target. We may construct a predictor using an LLM p via

$h(\boldsymbol{x}) \doteq \texttt{predict}(p(\cdot \mid \boldsymbol{x}))$. Here, `predict` is some (task-specific) *prediction function* mapping LLM output strings to targets; e.g., we may employ a standard decoding algorithm (Ficler & Goldberg, 2017; Fan et al., 2018; Holtzman et al., 2020), followed by stripping non-numeric tokens.

Model routing. Model routing is a means for achieving inference efficiency by sparingly calling larger models only on "hard" prompts. More precisely, suppose we have $M \geq 2$ LLMs $\{p^{(m)}\}_{m\in[M]}$, with corresponding *inference costs* $\{c^{(m)}\}_{m\in[M]}$ denoting, e.g., their average monetary costs for processing one prompt. We assume $c^{(1)} \leq c^{(2)} \leq \ldots \leq c^{(M)}$. Let $h^{(m)}(\boldsymbol{x}) \doteq \texttt{predict}(p^{(m)}(\cdot \mid \boldsymbol{x}))$ denote the predictor for the m^{th} LLM. Let $r\colon \mathcal{X} \to [M]$ be a *router* that, given a prompt, predicts the most suitable LLM. The standard ("static") routing problem seeks to solve (cf. Chen et al. (2023b); Dekoninck et al. (2025); Woisetschläger et al. (2025); Somerstep et al. (2025))

$$\min_{r\colon \mathcal{X}\to[M]} \sum_{m\in[M]} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})}\left[\mathbf{1}(r(\boldsymbol{x})=m)\cdot \ell(\boldsymbol{x},\boldsymbol{y},h^{(m)})\right] : \sum_{m\in[M]} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})}\left[\mathbf{1}(r(\boldsymbol{x})=m)\cdot c^{(m)}\right] \leq B. \quad (1)$$

Here, $B \geq 0$ is some fixed budget on the cost of the routed solution.

Model routing strategies. The most naïve routing strategy *randomly* assigns prompts to the various models, potentially pruning inadmissible solutions (see Appendix D). A more refined strategy is to *learn* a router (see Appendix I). Hu et al. (2024b) (see also Narasimhan et al. (2022)), proposed to construct a predictor $\gamma^{(m)}\colon \mathcal{X} \to \mathbb{R}_+$ of the expected loss incurred by each LLM $h^{(m)}$, and route via

$$r(\boldsymbol{x}) = \operatorname*{argmin}_{m\in[M]} \left[\gamma^{(m)}(\boldsymbol{x}) + \lambda \cdot c^{(m)}\right]. \quad (2)$$

Here, $\lambda \geq 0$ is a hyper-parameter trading between cost and quality. In §5, we formalise that (2) is a plug-in estimator of a theoretically optimal routing rule (cf. also Somerstep et al. (2025)). Given a training sample $S_{\text{tr}} \doteq \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{N_{\text{tr}}}$, there are several approaches to construct γ. For example, given a text embedder $\boldsymbol{\varphi}\colon \mathcal{X} \to \mathbb{R}^{D_{\text{P}}}$ (e.g., BERT (Devlin et al., 2019), Sentence-T5 (Ni et al., 2022)):

$$\gamma^{(m)}_{\text{lin}}(\boldsymbol{x}) = \mathbf{w}_m^\top \boldsymbol{\varphi}(\boldsymbol{x}) + b_m, \quad (3)$$

where $\mathbf{w}_m \in \mathbb{R}^{D_{\text{P}}}, b_m \in \mathbb{R}$ and (optionally) $\boldsymbol{\varphi}$ may be fit with a suitable empirical loss such as the mean squared error. A related matrix factorisation approach operating over *frozen* embeddings was proposed in Ong et al. (2025), fitted on either pairwise (Ong et al., 2025) or pointwise (Zhao et al., 2024a) supervision. Alternatively, we may use a K-NN estimator (Hu et al., 2024b, Section 5.1):

$$\gamma^{(m)}_{\text{kNN}}(\boldsymbol{x}) = \frac{1}{k} \sum_{i\in \text{NN}(\boldsymbol{x},k)} \ell(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}, h^{(m)}), \quad (4)$$

where $\text{NN}(\boldsymbol{x}, k)$ denotes the k-nearest neighbours of (the embeddings of) $\boldsymbol{x}$ in S_{tr}.

3 Model Routing with a Dynamic LLM Pool

We now formalise the model routing problem when the set of LLMs may vary *dynamically*.

Routing with a dynamic pool: motivation. The routing setup in (1) assumed a *static* pool of LLMs: indeed, (3) only estimates parameters $\{(\mathbf{w}_m, b_m)\}_{m\in[M]}$ for a fixed pool of LLMs, namely, $\{p^{(m)}\}_{m\in[M]}$. We are interested in routing with a *dynamic* pool of LLMs. The importance of this setting is twofold. First, new models are frequently released, and old models can be deprecated. It is highly desirable for a router to reflect these changes: e.g., smaller LMs are especially released very frequently, and leveraging them is critical for effective routing in a low-cost regime. Second, even with a fixed set of candidate LLMs, the pool of LLMs available for routing at test-time can change dynamically. For example, routing API services may see changes owing to the availability of GPUs for serving, licensing requirements, and suitability to the user's particular task.

There are two natural strategies to handle such a dynamic LLM pool: (i) *reuse* an existing router trained against an older pool, or (ii) *retrain* a router with each change to the pool. While simple, neither approach is wholly satisfactory. The first approach can be practically undesirable: failing to leverage newer LLMs can result in a poor user experience for end-users of a routing system. The second approach can incur non-trivial overhead (cf. Appendix H): for every newly released LLM, one

needs to perform router retraining and re-deployment. Further, such retraining requires annotating labelled samples with each new LLM, which runs the risk of *overfitting* when using too few samples, or incurs a high *annotation cost* otherwise. These motivate the following alternate setup.

Routing with a dynamic pool: setup. Let $\mathcal{H}_{\text{all}}$ denote the set of all possible LLM predictors, where for simplicity we assume $|\mathcal{H}_{\text{all}}| < +\infty$. Let $\mathbb{H} \doteq 2^{\mathcal{H}_{\text{all}}}$ denote the set of all subsets of $\mathcal{H}_{\text{all}}$. Let $\mathcal{H}_{\text{tr}} = \{h_{\text{tr}}^{(1)}, \ldots, h_{\text{tr}}^{(M)}\} \in \mathbb{H}$ denote the set of LLM predictors observed during training. During evaluation, we seek to route amongst $\mathcal{H}_{\text{te}} = \{h_{\text{te}}^{(1)}, \ldots, h_{\text{te}}^{(N)}\} \in \mathbb{H}$. The original routing problem in (1) corresponds to $\mathcal{H}_{\text{tr}} = \mathcal{H}_{\text{te}}$. However, we aim to allow $\mathcal{H}_{\text{tr}} \neq \mathcal{H}_{\text{te}}$, including $\mathcal{H}_{\text{tr}} \cap \mathcal{H}_{\text{te}} = \varnothing$.

To accommodate such a dynamic LLM pool, we first modify our router to accept both a prompt *and* a set of candidate LLMs, with the goal to pick the best option from this set; i.e., we consider *dynamic routers* $\mathcal{R} \doteq \{r(\cdot, \mathcal{H}) \colon \mathcal{X} \to [|\mathcal{H}|] \mid \mathcal{H} \in \mathbb{H}\}$. Next, we assume that the set of training LLMs $\mathcal{H}_{\text{tr}}$ is itself drawn from some *meta-distribution* $\mathfrak{H}$ over $\mathbb{H}$. Rather than perform well on the *specific* set $\mathcal{H}_{\text{tr}}$, we would like to generalise to *any* set of LLMs drawn from $\mathfrak{H}$. Concretely, we wish to solve:

$$\min_{r \in \mathcal{R}} \mathbb{E}\left[\sum_{m \in [|\mathcal{H}|]} \mathbf{1}(r(\boldsymbol{x}, \mathcal{H}) = m) \cdot \ell(\boldsymbol{x}, \boldsymbol{y}, h^{(m)})\right] : \mathbb{E}\left[\sum_{m \in [|\mathcal{H}|]} \mathbf{1}(r(\boldsymbol{x}, \mathcal{H}) = m) \cdot c(h^{(m)})\right] \leq B, \tag{5}$$

where as before $B \geq 0$ denotes a cost budget, $\mathcal{H} \doteq \{h^{(1)}, \ldots, h^{(M)}\} \sim \mathfrak{H}$ denotes a sample of M LLMs, $c \colon \mathcal{H}_{\text{all}} \to \mathbb{R}_+$ denotes the cost of a given LLM, and $\mathbb{E}$ is a shorthand for $\mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}, \mathcal{H})}$.

Bayes-optimal routing with a dynamic pool. To guide the design of a dynamic router, we study the *Bayes-optimal* rule for (5). Interestingly, the Bayes-optimal rule *decomposes* across each of the constituent LLMs. The result is known for a *fixed* pool $\mathcal{H} \in \mathbb{H}$ (Jitkrittum et al., 2023, Lemma F.1), (Dekoninck et al., 2025; Somerstep et al., 2025). The distinction arises for a *varying* pool, where the result closely mirrors (Tailor et al., 2024, Eq. 6), derived in a related setting (see Appendix I).

Proposition 1 (Optimal dynamic routing)**.** *Under a mild regularity condition on $\mathbb{P}$, for any input $\boldsymbol{x} \in \mathcal{X}$, LLM pool $\mathcal{H} \in \mathbb{H}$, and budget $B > 0$, there exists a Lagrange multiplier $\lambda_{\mathfrak{H}} \geq 0$ such that the optimal dynamic router r^* for the constrained optimization in (5) is*

$$r^*(\boldsymbol{x}, \mathcal{H}) = \operatorname*{argmin}_{m \in [|\mathcal{H}|]} \left[\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x}, \boldsymbol{y}, h^{(m)})\right] + \lambda_{\mathfrak{H}} \cdot c(h^{(m)})\right]. \tag{6}$$

The above uses standard Lagrangian analysis to convert the original *constrained* objective into an *unconstrained* one. Intuitively, it is optimal to route to the model with the lowest expected loss, after applying a *cost adjustment* of $\lambda_{\mathfrak{H}} \cdot c(h^{(m)})$ to the loss. The parameter $\lambda_{\mathfrak{H}} \geq 0$ allows one to trade off the expected quality and the average cost. If one wishes to *sweep* B to trace a cost-quality *deferral curve* (Appendix F.2), one may equally treat $\lambda_{\mathfrak{H}}$ as a tunable constant. See also Appendix E.

Consider a setting where an LLM response may be compared to a ground-truth target based on an exact match criteria: i.e., we pick the 0-1 loss $\ell(\boldsymbol{x}, \boldsymbol{y}, h^{(m)}) = \mathbf{1}[h^{(m)}(\boldsymbol{x}) \neq \boldsymbol{y}]$, with values either 0 (incorrect) or 1 (correct). Here, the optimal rule in (6) becomes:

$$r^*(\boldsymbol{x}, \mathcal{H}) = \operatorname*{argmin}_{m \in [|\mathcal{H}|]} \left[\gamma^*(\boldsymbol{x}, h^{(m)}) + \lambda_{\mathfrak{H}} \cdot c(h^{(m)})\right], \text{ where } \gamma^*(\boldsymbol{x}, h) \doteq \mathbb{P}\left[\boldsymbol{y} \neq h(\boldsymbol{x}) \mid \boldsymbol{x}\right]. \tag{7}$$

For simplicity, we will focus on the the 0-1 loss henceforth, and consider (7) as the optimal router; our results can be readily adapted (as we shall see empirically in §6) to other loss functions via (6).

Plug-in routing with a dynamic pool. The above suggests a simple router for a dynamic LLM pool $\mathcal{H}_{\text{te}} = \{h_{\text{te}}^{(n)}\}_{n \in [N]}$: construct a plug-in estimator $\gamma(\boldsymbol{x}, h)$ of $\gamma^*(\boldsymbol{x}, h)$, and estimate (7) via

$$r(\boldsymbol{x}, \mathcal{H}_{\text{te}}) = \operatorname*{argmin}_{n \in [N]} \left[\gamma(\boldsymbol{x}, h_{\text{te}}^{(n)}) + \lambda \cdot c(h_{\text{te}}^{(n)})\right]. \tag{8}$$

A key question arises: how should we parameterise $\gamma(\boldsymbol{x}, h)$? We study this question next.

UniRoute: Routing with a Dynamic LLM Pool

(1) **Train base router.** Fit parameters of γ_{uni} from (9) on training set $S_{\text{tr}} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{N_{\text{tr}}}$.

(2) **Embed test LLMs.** Compute $\boldsymbol{\Psi}(h_{\text{te}})$ for each test LLM $h_{\text{te}} \in \mathcal{H}_{\text{te}}$, e.g., via (10) on a validation set $S_{\text{val}} = \{(\boldsymbol{x}^{(j)}, \boldsymbol{y}^{(j)})\}_{j=1}^{N_{\text{val}}}$.

(3) **Route on new prompts.** Given a new input prompt $\boldsymbol{x}$, pick the best test LLM via (8).

4 UniRoute: Universal Routing via an LLM Feature Representation

To enable routing with a dynamic LLM pool, we propose to parameterise γ by representing both prompts *and* LLMs as feature vectors. Specifically, for fixed $K > 1$ let $\boldsymbol{\Phi} \colon \mathcal{X} \to \mathbb{R}^K$ and $\boldsymbol{\Psi} \colon \mathcal{H}_{\text{all}} \to \mathbb{R}^K$ denote *feature maps* for prompts and LLMs respectively. Then, we propose:

$$\gamma_{\text{uni}}(\boldsymbol{x}, h) = \boldsymbol{\Phi}(\boldsymbol{x})^\top \boldsymbol{\Psi}(h). \tag{9}$$

We may now fit any parameters associated with $\boldsymbol{\Phi}, \boldsymbol{\Psi}$ on the training set S_{tr}, and then route as before via (8). Crucially, provided we define an easily computable $\boldsymbol{\Psi}$, this seamlessly handles any $h \in \mathcal{H}_{\text{all}}$, *including one unobserved during training*; this is analogous to semantic output codes for zero-shot classification (Palatucci et al., 2009). Thus, (9) provides an approach for *universal routing* with dynamic LLM pools. The bilinear form is a natural generalisation of the standard model identity based routing, wherein one could interpret the model embedding as a one-hot vector. To *realise* the potential of this approach, it remains to specify $\boldsymbol{\Phi}(\boldsymbol{x})$ and $\boldsymbol{\Psi}(h)$.

Prompt representation $\boldsymbol{\Phi}$. Good choices for $\boldsymbol{\Phi}(\boldsymbol{x}) \in \mathbb{R}^K$ have been well-studied in prior work (Hu et al., 2024b). A standard choice is to leverage (a linear projection of) a general-purpose text embedding such as OpenAI (2025); Lee et al. (2025); Wang et al. (2024b); Lee et al. (2024b).

LLM representation $\boldsymbol{\Psi}$. A good choice for $\boldsymbol{\Psi}(h)$ is less apparent than that for $\boldsymbol{\Phi}(\boldsymbol{x})$. Observe that a linear router (3) for a static pool $\mathcal{H}_{\text{tr}} = \{h_{\text{tr}}^{(1)}, \ldots, h_{\text{tr}}^{(M)}\}$ corresponds to a *one-hot* LLM representation $\boldsymbol{\Psi}_{\text{oh}}(h) = \left[\mathbf{1}(h = h_{\text{tr}}^{(m)})\right]_{m \in [M]}$, which is inherently tied to the LLM pool $\mathcal{H}_{\text{tr}}$. We now examine an alternative representation based on an LLM's performance on a subset of prompts.

4.1 Representing an LLM via the Prediction Error Vector

A key desideratum for is that $\boldsymbol{\Psi}(h)^\top \boldsymbol{\Psi}(h')$ ought to be large for a pair (h, h') of "similar" LLMs, and small for a pair of "dissimilar" LLMs. We posit that two LLMs are "similar" if they *have comparable performance on a validation prompt set* (i.e., a set of prompts following the same distribution as those encountered in deployment). This is akin to proposals in Thrush et al. (2024); Zhuang et al. (2024).

Concretely, suppose that we have access to a small (labelled) validation set $S_{\text{val}} = \{(\boldsymbol{x}^{(j)}, \boldsymbol{y}^{(j)})\}_{j=1}^{N_{\text{val}}}$. Further, suppose that *any* LLM $h \in \mathcal{H}_{\text{all}}$ – *including new LLMs unobserved during training* – can be evaluated efficiently on these prompts. Then, one may represent the LLM based on its *prediction error vector* on prompts from S_{val}: for suitable $F \colon \mathbb{R}^{N_{\text{val}}} \to \mathbb{R}^K$ (e.g., a linear projection), we choose:

$$\boldsymbol{\Psi}(h) = F\left(\left[\mathbf{1}(\boldsymbol{y}^{(j)} \neq h(\boldsymbol{x}^{(j)}))\right]_{j \in [N_{\text{val}}]}\right) \in \mathbb{R}^K. \tag{10}$$

A special case of this is the K-NN method (4) from Hu et al. (2024b) applied to S_{val}: for $\boldsymbol{\Psi}_{\text{knn}}(h) = \left[\mathbf{1}(\boldsymbol{y}^{(j)} \neq h(\boldsymbol{x}^{(j)}))\right]_{j \in [N_{\text{val}}]} \in \mathbb{R}^{N_{\text{val}}}$ and $\boldsymbol{\Phi}_{\text{knn}}(\boldsymbol{x}) \in \{0, 1\}^{N_{\text{val}}}$ indicating which validation samples are the k-nearest neighbours of $\boldsymbol{x}$, (9) exactly reduces to (4). In general, however, it can prove useful to parameterise F and *learn* some compressed LLM representation in $K \ll N_{\text{val}}$ dimensions.

4.2 Discussion: UniRoute versus Standard Routing

A central assumption in UniRoute is that for any new LLM, one may efficiently compute $\boldsymbol{\Psi}(\cdot)$; for $\boldsymbol{\Psi}(\cdot)$ given by (10), this in turn assumes that any new LLM can be efficiently evaluated on S_{val}. This

only requires *black-box access* to the LLMs in the pool (e.g., as offered by standard APIs); the LLM weights need not be exposed. We stress some further important points regarding this.

Choice of validation prompt set. The success of `UniRoute` critically requires that the validation prompts S_{val} reasonably reflect those encountered during deployment. In our experiments, we largely take S_{val} to be a random subset of the training set S_{tr}: since the core assumption of training a router on S_{tr} is that these prompts reasonably reflect test prompts, a random subset can also be assumed to mimic test prompts. More generally, the validation prompts could be hand curated based on domain knowledge, or drawn from a standard benchmark suite (which is often available for any new LLM).

Comparison to router retraining. The assumption that each new LLM can be evaluated on S_{val} raises a natural question: why not simply retrain a conventional (static) router on S_{val}? This approach has two drawbacks. First, such retraining is susceptible to overfitting; this is because S_{val} *is assumed to be of modest size* (e.g., $\mathcal{O}(10^3)$), such that performing inference for a new LLM on S_{val} is not prohibitive. Consequently, `UniRoute` can thus yield better quality (as we shall demonstrate empirically). Second, router retraining and redeployment imposes additional overhead (e.g., engineering effort) compared to `UniRoute`. Further to these, `UniRoute` offers a more flexible approach that can handle bespoke routing use-cases, such as routing amongst dynamic model *subsets*; see Appendix H for more discussion.

Comparison to K-NN. K-NN routing (Hu et al., 2024b) – which, as noted in §4.1, is a special case of `UniRoute` – *does* support new LLMs without retraining. Indeed, one may readily compute γ_{kNN} in (4) based solely on the prediction error vector. However, as with naïve retraining, this approach may not generalise favourably as S_{val} is assumed to be of modest size; indeed, even in moderate data regimes, Zhuang et al. (2024) observed that K-NN may underperform compared to their proposed Matrix Factorization approach ("MatFac" in our experiments). Further, it does not exploit any information from the (potentially large) *training* set S_{tr}. Nonetheless, K-NN offers a simple means of making non-linear routing decisions. We now study alternate means of leveraging non-linearity in $\boldsymbol{\Psi}$.

5 `UniRoute` WITH CLUSTER-BASED LLM FEATURE REPRESENTATIONS

Building on the above, we now consider certain *cluster-based* LLM representations.

5.1 REPRESENTING AN LLM VIA PER-CLUSTER PREDICTION ERRORS

We propose an instantiation of (10) that represents any LLM $h \in \mathcal{H}_{\text{all}}$ through its average errors $\boldsymbol{\Psi}_{\text{clust}}(h) \in [0,1]^K$ on $K > 1$ pre-defined *clusters*. Similarly, we represent prompts via their cluster membership $\boldsymbol{\Phi}_{\text{clust}}(\boldsymbol{x}) \in \{0,1\}^K$. This yields the following instantiation of (9):

$$\gamma_{\text{clust}}(\boldsymbol{x}, h) \doteq \boldsymbol{\Phi}_{\text{clust}}(\boldsymbol{x})^\top \boldsymbol{\Psi}_{\text{clust}}(h). \tag{11}$$

Intuitively, $\gamma_{\text{clust}}(\boldsymbol{x}, h)$ estimates the performance of a given LLM on a prompt $\boldsymbol{x}$ by examining its performance on *similar* prompts, i.e., those belonging to the same cluster.

Naïvely, one may directly cluster S_{val}; however, this is prone to overfitting, since (by assumption) the set is of modest size. Thus, we instead cluster the prompts in the *training* set S_{tr}. We then use this to group the *validation* prompts into K disjoint clusters, and compute per-cluster errors for a new LM using S_{val}. Concretely, given a text embedder $\boldsymbol{\varphi} \colon \mathcal{X} \to \mathbb{R}^{D_{\text{P}}}$, we compute $\boldsymbol{\Phi}_{\text{clust}}(\boldsymbol{x}), \boldsymbol{\Psi}_{\text{clust}}(h)$ via:

(i) Cluster the *training* set embeddings $\{\boldsymbol{\varphi}(\boldsymbol{x}^{(i)})\}_{i=1}^{N_{\text{tr}}}$ to construct K non-overlapping clusters. This yields a cluster assignment map $\boldsymbol{\Phi}_{\text{clust}} \colon \mathcal{X} \to \{0,1\}^K$, where the k-th index is 1 when $\boldsymbol{x}$ belongs to cluster k (i.e., it is on average closest to samples in cluster k). This step does not require labels.

(ii) Assign each *validation* set prompt to a cluster. Let $C_k \doteq \{(\boldsymbol{x}, \boldsymbol{y}) : (\boldsymbol{x}, \boldsymbol{y}) \in S_{\text{val}}, \boldsymbol{\Phi}_{\text{clust},k}(\boldsymbol{x}) = 1\}$ be the subset of the validation set that belongs to cluster k.

(iii) For any LLM $h \in \mathcal{H}_{\text{all}}$, compute $\boldsymbol{\Psi}_{\text{clust}}(h) \in [0,1]^K$ using its per-cluster validation errors:

$$\Psi_{\text{clust},k}(h) \doteq \frac{1}{|C_k|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in C_k} \mathbf{1}\big[\boldsymbol{y} \neq h(\boldsymbol{x})\big]. \tag{12}$$

Plugging these into (11), we may now approximate the expected loss for $h_{\text{te}}^{(n)}$ on an input prompt $\boldsymbol{x}$ using the average error of the LLM on the cluster the prompt is assigned to, and route via (8).

This cluster-based instantiation of `UniRoute` can route with new LLMs in a highly efficient manner: given any new test LLM, we simply need to estimate its average per-cluster clusters for all validation prompts. This does *not* require any expensive gradient updates, and is a *one-off cost*: further routing can operate entirely on this vector, and is oblivious to any further changes in the LLM pool.

A natural choice of clustering algorithm in step (ii) is K-means (MacQueen, 1967). For $K = 1$, the router devolves to the *ZeroRouter* from (Hu et al., 2024b) (see Appendix D). Empirically, we find `UniRoute` to be reasonably robust to the choice of K (Appendix G.2).

5.2 From Fixed to Learned Cluster Assignment Maps

To improve the above cluster-based router, we may further seek to exploit the labels in S_{tr}: given the same clustering, we propose to *learn* a cluster assignment map $\boldsymbol{\Phi}_{\text{clust}}(\cdot;\boldsymbol{\theta}) \in [0,1]^K$ that can better map a prompt to a distribution over clusters. Specifically, we set $\boldsymbol{\Phi}_{\text{clust},k}(\boldsymbol{x};\boldsymbol{\theta}) \propto \exp\left(\theta_k^\top \boldsymbol{\varphi}(\boldsymbol{x})\right)$ for parameters $\boldsymbol{\theta} \in \mathbb{R}^{K \times D_{\text{P}}}$ and text embedding $\boldsymbol{\varphi}$. Analogous to (11), we have

$$\gamma_{\text{clust}}(\boldsymbol{x}, h; \boldsymbol{\theta}) = \boldsymbol{\Phi}_{\text{clust}}(\boldsymbol{x};\boldsymbol{\theta})^\top \boldsymbol{\Psi}_{\text{clust}}(h),$$

where, as in (12), $\boldsymbol{\Psi}_{\text{clust}}(h)$ denotes the per-cluster errors estimated from the validation set S_{val}. To pick $\boldsymbol{\theta}$, we minimize the log loss on S_{tr} against the correctness labels for the training LMs $\mathcal{H}_{\text{tr}}$:

$$-\sum_{(\boldsymbol{x},\boldsymbol{y})\in S_{\text{tr}}} \sum_{h\in\mathcal{H}_{\text{tr}}} \mathbf{1}\left[\boldsymbol{y} \neq h(\boldsymbol{x})\right] \cdot \log\, \gamma_{\text{clust}}\left(\boldsymbol{x}, h; \boldsymbol{\theta}\right) + \mathbf{1}\left[\boldsymbol{y} = h(\boldsymbol{x})\right] \cdot \log\left(1 - \gamma_{\text{clust}}\left(\boldsymbol{x}, h; \boldsymbol{\theta}\right)\right).$$

5.3 Excess Risk Bound

We now present an excess risk bound for our cluster-based routing strategy. Suppose we represent the underlying data distribution over $(\boldsymbol{x}, \boldsymbol{y})$ by a mixture of K latent components: $\mathbb{P}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{k=1}^K \pi_k \cdot \mathbb{P}(\boldsymbol{x}, \boldsymbol{y} \,|\, z = k)$, where z denotes a latent random variable that identifies the mixture component and $\pi_k = \mathbb{P}(z = k)$. For a fixed k, we may denote the probability of incorrect predictions for $h \in \mathcal{H}$ conditioned on $z = k$ by $\Psi_k^*(h) \doteq \mathbb{P}_{\boldsymbol{x},\boldsymbol{y}|z=k}\left[h(\boldsymbol{x}) \neq \boldsymbol{y}\right]$ .Then, cluster-based routing seeks to mimic the following rule:

$$\tilde{r}^*(\boldsymbol{x}, \mathcal{H}_{\text{te}}) = \operatorname*{argmin}_{n\in[N]} \left[\sum_{k\in[K]} \mathbb{P}(z = k|\boldsymbol{x}) \cdot \Psi_k^*(h_{\text{te}}^{(n)}) + \lambda \cdot c(h_{\text{te}}^{(n)})\right]. \tag{13}$$

Proposition 2. *Let r^* be as per (7). For any $\mathcal{H}_{\text{te}} = \{h_{\text{te}}^{(n)}\}_{n\in[N]} \in \mathbb{H}$, $h_{\text{te}}^{(n)} \in \mathcal{H}_{\text{te}}$, and $\boldsymbol{x} \in \mathcal{X}$, let: $\Delta_k(\boldsymbol{x}, h_{\text{te}}^{(n)}) \doteq \left|\mathbb{P}_{\boldsymbol{y}|\boldsymbol{x},z=k}\left[h_{\text{te}}^{(n)}(\boldsymbol{x}) \neq \boldsymbol{y}\right] - \Psi_k^*(h_{\text{te}}^{(n)})\right|$. Let $R_{01}(r, \mathcal{H}_{\text{te}}) \doteq \sum_n \mathbb{P}\left[h_{\text{te}}^{(n)}(\boldsymbol{x}) \neq \boldsymbol{y} \wedge r(\boldsymbol{x}, \mathcal{H}_{\text{te}}) = n\right]$ denote the 0-1 risk. Then under a regularity condition on $\mathbb{P}$,*

$$\mathbb{E}_{\mathcal{H}_{\text{te}}}\left[R_{01}(\tilde{r}^*, \mathcal{H}_{\text{te}})\right] - \mathbb{E}_{\mathcal{H}_{\text{te}}}\left[R_{01}(r^*, \mathcal{H}_{\text{te}})\right] \leq \mathbb{E}_{\mathcal{H}_{\text{te}},\boldsymbol{x}}\left[\max_{h_{\text{te}}^{(n)}\in\mathcal{H}_{\text{te}},k\in[K]} \Delta_k(\boldsymbol{x}, h_{\text{te}}^{(n)})\right].$$

This suggests that the quality gap between cluster-based routing in (13) and the optimal rule in (7) is bounded by the discrepancy between the per-cluster and per-prompt errors: i.e., the gap between the LLM's error on a prompt versus the average constituent cluster error (see Appendix C.3 for proof).

5.4 Discussion: `UniRoute` versus existing work

`UniRoute` allows a trained router to make use of new test-time LLMs without retraining. This is unlike most, though not all existing approaches; indeed, recent works Feng et al. (2024); Zhao et al. (2024b); Li (2025) have also proposed dynamic routing approaches. In comparison, we provide a formally-grounded approach that avoids dependence on exogenous *task label* information for prompts, and relies on established statistical learning primitives.

More precisely, Li (2025) also proposes the use of a certain LLM embedding vector, and is thus the most related work. However, there are important distinctions to `UniRoute`. First, Li (2025) rely on RL-based *policy gradient* optimisation; the significant recent advances in RL notwithstanding, this raises the risk of training instability, and computational burden. By contrast, we consider standard statistical learning (akin to most recent routing works, e.g., Somerstep et al. (2025)), and rely on standard gradient descent. Second, the LLM embeddings in Li (2025) depend on *other* LLMs in the pool, as well as the *order* in which the LLMs are added. By contrast, our embeddings are *pool agnostic*, and highly intuitive (per-cluster errors). Third, to construct the LLM embeddings, Li (2025) perform an iterative optimisation of a variational objective, which further relies on estimation of prompt difficulty. By contrast, our correctness-vector based representation requires a simple calculation of per-cluster errors. We defer a full discussion to Appendix I.

6 Experimental Results

We demonstrate the effectiveness of `UniRoute` when observing **new LLMs at test time**.

Experimental setup. We present results on EmbedLLM (Zhuang et al., 2024), RouterBench (Hu et al., 2024b), a Math+Code dataset from Dekoninck et al. (2025) (containing a subset of Minerva Math and LiveCodeBench (Lewkowycz et al., 2022; Jain et al., 2024)), SPROUT o3-mini (Somerstep et al., 2025), Headlines (Sinha & Khandait, 2021; Chen et al., 2023b), OpenLLMv2 (Fourrier et al., 2024), and Chatbot Arena (Zheng et al., 2023). All datasets use binary accuracy as the evaluation metric. Thus, all methods rely on the deferral rule described in (7).

LLM pool construction. We partition the set of LLMs into two disjoint sets: training models ($\mathcal{H}_{\mathrm{tr}}$ in §5) and testing models ($\mathcal{H}_{\mathrm{te}}$). For EmbedLLM (112 LLMs) and SPROUT o3-mini (15 LLMs), we use a random subset of $2/3$ for training and $1/3$ for testing. For Headlines, we hold out six LLMs for training and use the other six LLMs for testing (details in Appendix G.3). For RouterBench (11 LLMs in total), we use a random 50% for training and the rest for testing. See Appendix G.2 for details on hyper-parameter selection.

Example construction. Over 400 independent trials, for EmbedLLM, we randomly split examples into 60%/10%/30% for training, validation, and testing (for SPROUT o3-mini, Headlines, and RouterBench, we use 400 examples for validation). The training portion is for training a router, and only has correctness labels of training models. The validation split is the small dataset used to represent each test-time LLM as a feature vector (see §5.1). All baselines are evaluated on the test examples and *only* on the test LLMs. We use Gecko 1B (Lee et al., 2024b) (frozen) to produce a 768-dimensional prompt embedding for all methods.

Evaluation. We evaluate each method with a *deferral curve* (Appendix F.2; (Jitkrittum et al., 2023; Wang et al., 2024a; Hu et al., 2024b)), which plots the trade-off of average quality against cost. The trade-off is realized by varying $\lambda_{\mathfrak{H}}$ in (6). We report two evaluation metrics: (i) the quality-neutral cost (QNC), or the minimum relative cost needed to match the quality of the most accurate test LLM; and (ii) the area under the deferral curve (Area). The QNC is analogous to the call-performance threshold in Ong et al. (2025). For EmbedLLM, we use the number of parameters of the LLM as a proxy for the cost of processing one prompt. For SPROUT o3-mini, HeadLines, and RouterBench, we use each LLMs' API calling costs (in USD) as provided in the datasets.

We evaluate `UniRoute` cluster-based routing – using both the unsupervised and supervised cluster assignment maps (§5.1, §5.2) – against certain dynamic & static routing baselines, described next.

Baselines: dynamic routing. There are two simple, but surprisingly strong dynamic routing baselines:

- *ZeroRouter* (Hu et al., 2024b). A router that *randomizes* between two LLMs, chosen from those attaining a certain Pareto-optimal performance on the validation sample (details in Appendix D).
- *K-NN* (Hu et al., 2024b; Shnitzer et al., 2023). A special case of `UniRoute` (see §4.1) that for each prompt, looks up the K nearest prompts in the validation set in the space of prompt embeddings, computes γ for each test LLM using (4) (with the 0-1 loss), and routes via (2).

LLMBandit (Li, 2025) is also a natural baseline per §5.4. Unfortunately, an exact replication of Li (2025) is challenging, owing to the lack of a public implementation, coupled with their

Method \ Dataset	EmbedLLM		SPROUT o3-mini		Headlines		RouterBench	
	QNC ↓	Area ↑	QNC ↓	Area ↑	QNC ↓	Area ↑	QNC ↓	Area ↑
ZeroRouter (Hu et al., 2024b)	87.5% *	.607 *	100.0% *	.820 *	88.0% *	.819 *	99.9%	.689 *
K-NN (Hu et al., 2024b; Shnitzer et al., 2023)	45.9% *	.636 *	29.6% *	.844 *	43.7% *	.830 *	99.7%	.707 *
Retrained MLP (Hu et al., 2024b)	35.9% *	.641 *	80.9% *	.829 *	74.2% *	.823 *	99.2%	.710 *
Retrained MatFac (Ong et al., 2025; Zhuang et al., 2024)	36.6% *	.640 *	84.2% *	.825 *	80.9% *	.821 *	99.3%	.708 *
`UniRoute` (K-Means)	33.7%	.649 *	19.6%	.850	56.9% *	.828 *	99.4%	.712
`UniRoute` (LearnedMap)	33.1%	.652	23.4%	.846	34.9%	.832	99.6%	.711

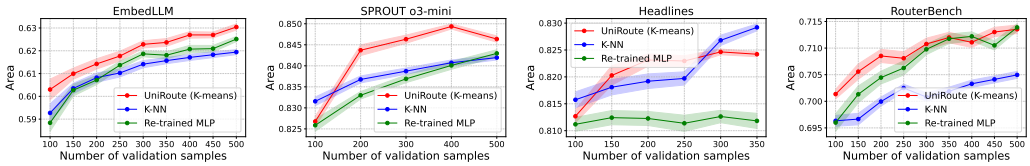


Figure 2: **Top**: We report the Quality-Neutral Cost (QNC), i.e., the minimum relative cost to achieve the same performance as the most accurate LLM, and the *area* under the deferral curve. * indicates the method is statistically significantly worse than `UniRoute` (LearnedMap) at significance level $\alpha = 0.01$. **Bottom**: Areas under the deferral curve (↑) with 96% CI on **unseen test LLMs** for varying validation sample sizes. `UniRoute` (K-means) is often better than the baselines across sample sizes. The re-trained MLP underperforms on two datasets due to overfitting to the smaller validation samples.

approach involving many non-trivial components (e.g., policy gradient training with a replay buffer). Nonetheless, Appendix I.1 presents a (favourable) comparison against the paper's quoted results.

Baselines: static routing. We reiterate that ***most existing routers in the literature are inherently tied to the LLMs they are trained with***; thus, such static routers ***are impractical*** in the dynamic routing setting. This is true of the multi-layer perceptron (MLP) (Hu et al., 2024b), matrix factorization (Ong et al., 2025; Zhuang et al., 2024) and BERT-based routers (Ong et al., 2025; Ding et al., 2024; Chen et al., 2024b; Mei et al., 2025), all of which have a *fixed* number of outputs (one per LLM). Despite this, we include two representative methods for completeness:

- ***Retrained** MLP & MatrixFac*: An MLP router (Hu et al., 2024b) and a factorised router (Ong et al., 2025; Zhuang et al., 2024), with one output per test LLM, trained on prompt embeddings to estimate γ. For each new LLM, these routers need to be retrained with an additional output using the small labeled validation sample available for the new LLM. These baselines are impractical as they are prone to *overfitting* to the small validation sample, and incur *frequent retraining* costs.

We excluded static routers with more complex architectures (e.g., Ding et al. (2024)) because they are even more susceptible to overfitting and less practical to retrain frequently.

Experimental results. We present deferral curves on EmbedLLM in Figure 3, and on other datasets in Appendix G.3. Each isolated × represents the cost and average test accuracy of one test LLM. The table in Figure 2 summarizes Area and QNC for four datasets; additional results in Appendix G.3.

`UniRoute` **is effective under dynamic LLM pools**. `UniRoute` yields competitive quality-cost trade-offs, with the gains over baselines being *statistically significant*. In particular, on EmbedLLM – featuring > 30 **unseen LLMs** – we provide compelling gains over K-NN. Further, on all datasets, we consistently outperform ZeroRouter, which was noted as a strong baseline in Hu et al. (2024b). We are also significantly better than the re-trained static routers on most datasets; as expected, these baselines underperform as a result of overfitting to the small validation set. Appendix G.4 further shows `UniRoute` is effective when the LLM representations are constructed from Chatbot Arena, and evaluated on *out-of-distribution* EmbedLLM prompts.

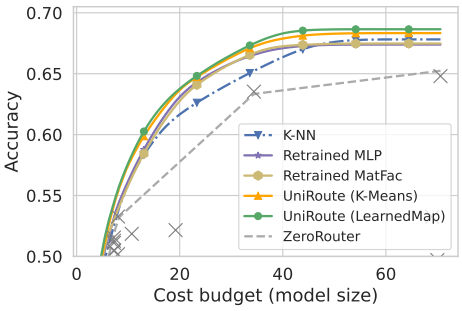


Figure 3: Deferral curves on EmbedLLM.

In Appendix I.1, we show that `UniRoute` performs favourably compared to results from Li (2025).

`UniRoute` **is effective even with a small validation sample**. We show in Figure 2 (bottom), that our proposal is often significantly better than K-NN across a range of validation sample sizes. One potential reason for this is the requirement in K-NN that only the retrieved neighbors from the validation set can be used to estimate test models' performance. It is hence unable to exploit the training set in any way. In contrast, our methods are able to exploit the training data either in an unsupervised (K-means) or supervised (LearnedMap) manner.

`UniRoute` **is robust to choice of clusters** K. Appendix G shows that in general, `UniRoute` is effective for several different K values, and our hyperparameter selector picks an effective K.

`UniRoute` **maintains effectiveness under static LLM pools**. While the dynamic pool setting is the focal point of our work, we show in Appendix G.5 that even in the *static* LLM pool setting, `UniRoute` typically performs comparable to most baselines.

`UniRoute`**'s LLM representations capture similar LLMs.** Appendix G.6 shows that similar LLMs are close in the space of cluster-based LLM embeddings of `UniRoute` (Section 5).

7 Limitations and Future Work

While `UniRoute` makes progress on the problem of dynamic routing, there are some limitations which present opportunities for future research. First, in the fully static LLM pool setting, `UniRoute` is not *guaranteed* to recover the performance of existing methods; designing *hybrid* static-dynamic routers that leverage the best of both paradigms would be of interest. Second, while we proposed certain natural cluster-based routers, there could be other instances of `UniRoute`; a more systematic study of this design space would be worthwhile. This may be particularly fruitful in settings without a clean cluster structure, wherein `UniRoute` (K-Means) is seen to underperform `UniRoute` (LearnedMap). Finally, while the choice of a random subset of training prompts for S_{val} is intuitive, further optimizing for a *diverse* subset (i.e., finding a coreset) can potentially give an even better representation; this is complementary to our current proposal, and is an interesting topic for future study.

Reproducibility Statement

We have made efforts to ensure reproducibility of our work. Our experiments on dynamic routing rely on training, validation, and test data splits. Each data split may contain training LLM, test LLMs, or both. These details are explained in Appendix F.1. The model architecture of our dynamic router is detailed in Appendix F.3. Tuning of hyper-parameters and parameter choices are clearly described in Appendix G.1, and Appendix G.2, respectively. All datasets used are publicly available and properly attributed. Data preprocessing steps are in Section 6 and Appendix G.3. For our theoretical results, the complete proofs are given in Appendix C, along with several auxiliary lemmas.

References

Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless Fowlkes, Stefano Soatto, and Pietro Perona. Task2vec: Task embedding for meta-learning. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6429–6438, 2019. doi: 10.1109/ICCV.2019.00653.

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3zKtaqxLhW.

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam . Automix: Automatically mixing language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=e6WrwIvgzX.

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark,

Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. PaLM 2 technical report, 2023.

Peter L. Bartlett and Marten H. Wegkamp. Classification with a reject option using a hinge loss. *Journal of Machine Learning Research*, 9(59):1823–1840, 2008. URL http://jmlr.org/papers/v9/bartlett08a.html.

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for fast test-time prediction. In *International Conference on Machine Learning*, 2017.

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.

Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. In Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen (eds.), *Computer Vision — ECCV 2002*, pp. 707–720, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-47969-7.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. In *International Conference on Machine Learning (ICML)*, July 2024a.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. MEDUSA: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024b.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. QuIP: 2-bit quantization of large language models with guarantees. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Boyuan Chen, Mingzhi Zhu, Brendan Dolan-Gavitt, Muhammad Shafique, and Siddharth Garg. Model cascading for code: Reducing inference costs with model cascading for LLM based code generation, 2024a. URL https://arxiv.org/abs/2405.15842.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023a.

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*, 2023b.

Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. RouterDC: Query-based router by dual contrastive learning for assembling large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b.

François Chollet et al. Keras. https://keras.io, 2015.

C Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on information theory*, 16(1):41–46, 1970.

Yu-Neng Chuang, Prathusha Kameswara Sarma, Parikshit Gopalan, John Boccio, Sara Bolouki, Xia Hu, and Helen Zhou. Learning to route LLMs with confidence tokens. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=Uo8mUogGDM.

Corinna Cortes, Giulia DeSalvo, and Mehryar Mohri. Learning with rejection. In *ALT*, 2016. URL https://cs.nyu.edu/~mohri/pub/rej.pdf.

Jordan Cotler, Kai Sheng Tai, Felipe Hernández, Blake Elias, and David Sussillo. Analyzing populations of neural networks via dynamical model embedding, 2023. URL https://arxiv.org/abs/2302.14078.

Christoph Dann, Yishay Mansour, Teodor Vanislavov Marinov, and Mehryar Mohri. Domain adaptation for robust model routing. In *Adaptive Foundation Models: Evolving AI for Personalized and Efficient Learning*, 2024. URL https://openreview.net/forum?id=86eGohKPyy.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

Jasper Dekoninck, Maximilian Baader, and Martin Vechev. A unified approach to routing and cascading for LLMs, 2025. URL https://arxiv.org/abs/2410.10347.

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, Minneapolis, Minnesota, June

2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Fnu Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit S Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham M. Kakade, Ali Farhadi, and Prateek Jain. MatFormer: Nested transformer for elastic inference. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=fYa6ezMxD5.

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. URL https://openreview.net/forum?id=02f3mUtqnM.

Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Adhiguna Kuncoro, Yani Donchev, Rachita Chhaparia, Ionel Gog, Marc'Aurelio Ranzato, Jiajun Shen, and Arthur Szlam. DiPaCo: Distributed path composition, 2024. URL https://arxiv.org/abs/2403.10616.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.681. URL http://dx.doi.org/10.18653/v1/2024.acl-long.681.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 889–898, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1082. URL https://aclanthology.org/P18-1082.

William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022. URL https://arxiv.org/abs/2101.03961.

Tao Feng, Yanzhen Shen, and Jiaxuan You. GraphRouter: A graph-based router for llm selections, 2024. URL https://arxiv.org/abs/2410.03834.

Jessica Ficler and Yoav Goldberg. Controlling linguistic style aspects in neural language generation. In *Proceedings of the Workshop on Stylistic Variation*, pp. 94–104, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4912. URL https://aclanthology.org/W17-4912.

Clémentine Fourrier, Nathan Habib, Alina Lozovskaya, Konrad Szafer, and Thomas Wolf. Open llm leaderboard v2. https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard, 2024.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10323–10337. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/frantar23a.html.

Yonatan Geifman and Ran El-Yaniv. SelectiveNet: A deep neural network with an integrated reject option. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2151–2159. PMLR, 09–15 Jun 2019.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle

Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish

Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The Llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Neel Guha, Mayee F Chen, Trevor Chow, Ishan S. Khare, and Christopher Re. Smoothie: Label free language model routing. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a. URL https://openreview.net/forum?id=pPSWHsgqRp.

Neel Guha, Mayee F Chen, Trevor Chow, Ishan S. Khare, and Christopher Re. Smoothie: Label free language model routing. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b. URL https://openreview.net/forum?id=pPSWHsgqRp.

Neha Gupta, Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Aditya Krishna Menon, and Sanjiv Kumar. Language model cascades: Token-level uncertainty and beyond. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KgaBScZ4VI.

Surya Narayanan Hari and Matt Thomson. Tryage: Real-time, intelligent routing of user prompts to large language models, 2023. URL https://arxiv.org/abs/2308.11601.

Amr Hendy, Mohamed Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla. How good are GPT models at machine translation? a comprehensive evaluation, 2023. URL https://arxiv.org/abs/2302.09210.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

Jinwu Hu, Yufeng Wang, Shuhai Zhang, Kai Zhou, Guohao Chen, Yu Hu, Bin Xiao, and Mingkui Tan. Dynamic ensemble reasoning for llm experts, 2024a. URL https://arxiv.org/abs/2412.07448.

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. In *Agentic Markets Workshop at ICML 2024*, 2024b. URL https://openreview.net/forum?id=IVXmV8Uxwh.

Zhongzhan Huang, Guoming Ling, Vincent S. Liang, Yupei Lin, Yandong Chen, Shanshan Zhong, Hefeng Wu, and Liang Lin. Routereval: A comprehensive benchmark for routing llms to explore model-level scaling up in LLMs, 2025. URL https://arxiv.org/abs/2503.10657.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. doi: 10.1162/neco.1991.3.1.79.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Joel Jang, Seungone Kim, Seonghyeon Ye, Doyoung Kim, Lajanugen Logeswaran, Moontae Lee, Kyungjae Lee, and Minjoon Seo. Exploring the benefits of training expert language models over instruction tuning. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-Blender: Ensembling large language models with pairwise ranking and generative fusion. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14165–14178, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.792. URL https://aclanthology.org/2023.acl-long.792/.

Wittawat Jitkrittum, Neha Gupta, Aditya Krishna Menon, Harikrishna Narasimhan, Ankit Singh Rawat, and Sanjiv Kumar. When does confidence-based cascade deferral suffice? In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=4KZhZJSPYU.

M.I. Jordan and R.A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, volume 2, pp. 1339–1344 vol.2, 1993. doi: 10.1109/IJCNN.1993.716791.

Anil Kag, Igor Fedorov, Aditya Gangrade, Paul Whatmough, and Venkatesh Saligrama. Efficient edge inference by selective query. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=jpR98ZdIm2q.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nv-embed: Improved techniques for training llms as generalist embedding models, 2025. URL https://arxiv.org/abs/2405.17428.

Chia-Hsuan Lee, Hao Cheng, and Mari Ostendorf. OrchestraLLM: Efficient orchestration of language models for dialogue state tracking. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1434–1445, Mexico City, Mexico, June 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.79. URL https://aclanthology.org/2024.naacl-long.79/.

Jinhyuk Lee, Zhuyun Dai, Xiaoqi Ren, Blair Chen, Daniel Cer, Jeremy R. Cole, Kai Hui, Michael Boratko, Rajvi Kapadia, Wen Ding, Yi Luan, Sai Meher Karthik Duddu, Gustavo Hernandez Abrego, Weiqiang Shi, Nithi Gupta, Aditya Kusupati, Prateek Jain, Siddhartha Reddy Jonnalagadda, Ming-Wei Chang, and Iftekhar Naim. Gecko: Versatile text embeddings distilled from large language models, 2024b. URL https://arxiv.org/abs/2403.20327.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. Llm inference serving: Survey of recent advances and opportunities, 2024a. URL https://arxiv.org/abs/2407.12391.

Tianle Li, Wei-Lin Chiang, and Lisa Dunlap. Introducing hard prompts category in Chatbot Arena. https://lmsys.org/blog/2024-05-17-category-hard, 2024b.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline, 2024c. URL https://arxiv.org/abs/2406.11939.

Yang Li. LLM Bandit: Cost-efficient llm generation via preference-conditioned dynamic routing, 2025. URL https://arxiv.org/abs/2502.02743.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024d.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432, Miami, Florida, USA, November 2024e. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.422. URL https://aclanthology.org/2024.emnlp-main.422/.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. Routing to the expert: Efficient reward-guided ensemble of large language models. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1964–1974, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.109. URL https://aclanthology.org/2024.naacl-long.109/.

J. MacQueen. Some methods for classification and analysis of multivariate observations. Proc. 5th Berkeley Symp. Math. Stat. Probab., Univ. Calif. 1965/66, 1, 281-297 (1967)., 1967.

David Madras, Toniann Pitassi, and Richard Zemel. Predict responsibly: Improving fairness and accuracy by learning to defer. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NeurIPS'18, pp. 6150–6160, Red Hook, NY, USA, 2018. Curran Associates Inc.

Anqi Mao, Christopher Mohri, Mehryar Mohri, and Yutao Zhong. Two-stage learning to defer with multiple experts. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024a. Curran Associates Inc.

Anqi Mao, Mehryar Mohri, and Yutao Zhong. Principled approaches for learning to defer with multiple experts. In Reneta P. Barneva, Valentin E. Brimkov, Claudio Gentile, and Aldo Pacchiano (eds.), *Artificial Intelligence and Image Analysis*, pp. 107–135, Cham, 2024b. Springer Nature Switzerland. ISBN 978-3-031-63735-3.

Anqi Mao, Mehryar Mohri, and Yutao Zhong. Realizable h-consistent and Bayes-consistent loss functions for learning to defer. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024c. URL https://openreview.net/forum?id=OcO2XakUUK.

Anqi Mao, Mehryar Mohri, and Yutao Zhong. Regression with multi-expert deferral. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 34738–34759. PMLR, 21–27 Jul 2024d. URL https://proceedings.mlr.press/v235/mao24d.html.

Kai Mei, Wujiang Xu, Shuhang Lin, and Yongfeng Zhang. Omnirouter: Budget and performance controllable multi-llm routing. *arXiv preprint arXiv:2502.20576*, 2025.

Alireza Mohammadshahi, Arshad Rafiq Shaikh, and Majid Yazdani. Routoo: Learning to route to large language models effectively, 2024.

Yannis Montreuil, Axel Carlier, Lai Xing Ng, and Wei Tsang Ooi. Adversarial robustness in two-stage learning-to-defer: Algorithms and guarantees, 2025. URL https://arxiv.org/abs/2502.01027.

Hussein Mozannar and David Sontag. Consistent estimators for learning to defer to an expert. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 7076–7087. PMLR, 13–18 Jul 2020.

Harikrishna Narasimhan, Wittawat Jitkrittum, Aditya Krishna Menon, Ankit Singh Rawat, and Sanjiv Kumar. Post-hoc estimators for learning to defer to an expert. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_jg6Sf6tuF7.

Harikrishna Narasimhan, Aditya Krishna Menon, Wittawat Jitkrittum, Neha Gupta, and Sanjiv Kumar. Learning to reject for balanced error and beyond. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=ta26LtNq2r.

Harikrishna Narasimhan, Aditya Krishna Menon, Wittawat Jitkrittum, and Sanjiv Kumar. Plugin estimators for selective classification with out-of-distribution detection. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=DASh78rJ7g.

Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Seungyeon Kim, Neha Gupta, Aditya Krishna Menon, and Sanjiv Kumar. Faster cascades via speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=vo9t20wsmd.

Jerzy Neyman and Egon Sharpe Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 231(694-706):289–337, 1933.

Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith Hall, Daniel Cer, and Yinfei Yang. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 1864–1874, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.146. URL https://aclanthology.org/2022.findings-acl.146/.

Lunyiu Nie, Zhimin Ding, Erdong Hu, Christopher Jermaine, and Swarat Chaudhuri. Online cascade learning for efficient inference over streams. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *The Thirteenth International Conference on Learning Representations*, 2025.

OpenAI. New embedding models and API updates. https://openai.com/index/new-embedding-models-and-api-updates/, 2025.

Mark Palatucci, Dean Pomerleau, Geoffrey Hinton, and Tom M. Mitchell. Zero-shot learning with semantic output codes. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems*, NIPS'09, pp. 1410–1418, Red Hook, NY, USA, 2009. Curran Associates Inc. ISBN 9781615679119.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *MLSys*, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/c4be71ab8d24cdfb45e3d06dbfca2780-Abstract-mlsys2023.html.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf, 2018.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

Mathieu Ravaut, Shafiq Joty, and Nancy F Chen. Summareranker: A multi-task mixture-of-experts re-ranking framework for abstractive summarization. *arXiv preprint arXiv:2203.06569*, 2022.

Ankit Singh Rawat, Manzil Zaheer, Aditya Krishna Menon, Amr Ahmed, and Sanjiv Kumar. When in doubt, summon the titans: Efficient inference with large models. *arXiv preprint arXiv:2110.10305*, 2021.

Ankit Singh Rawat, Veeranjaneyulu Sadhanala, Afshin Rostamizadeh, Ayan Chakrabarti, Wittawat Jitkrittum, Vladimir Feinberg, Seungyeon Kim, Hrayr Harutyunyan, Nikunj Saunshi, Zachary Nado, Rakesh Shivanna, Sashank J. Reddi, Aditya Krishna Menon, Rohan Anil, and Sanjiv Kumar. A little help goes a long way: Efficient LLM training by leveraging small LMs, 2024. URL https://arxiv.org/abs/2410.18779.

Aishwarya P S, Pranav Ajit Nair, Yashas Samaga, Toby Boyd, Sanjiv Kumar, Prateek Jain, and Praneeth Netrapalli. Tandem Transformers for inference efficient LLMs, 2024. URL https://arxiv.org/abs/2402.08644.

Sara Sangalli, Ertunc Erdil, and Ender Konukoglu. Expert load matters: operating networks at high accuracy and low manual effort. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Y2VQWfi7Vc.

Simone Scardapane, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. Why should we add early exits to neural networks? *Cognitive Computation*, 12:954–966, 2020.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=uLYc4L3C81A.

Avital Shafran, Roei Schuster, Thomas Ristenpart, and Vitaly Shmatikov. Rerouting LLM routers, 2025. URL https://arxiv.org/abs/2501.01818.

Tal Shnitzer, Anthony Ou, Mírian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. Large language model routing with benchmark datasets, 2023. URL https://arxiv.org/abs/2309.15789.

Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 22045–22055. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/fb2697869f56484404c8ceee2985b01d-Paper.pdf.

Ankur Sinha and Tanmay Khandait. Impact of news on the commodity market: Dataset and results. In Kohei Arai (ed.), *Advances in Information and Communication*, pp. 589–601, Cham, 2021. Springer International Publishing. ISBN 978-3-030-73103-8.

Seamus Somerstep, Felipe Maia Polo, Allysson Flavio Melo de Oliveira, Prattyush Mangal, Mírian Silva, Onkar Bhardwaj, Mikhail Yurochkin, and Subha Maity. Carrot: A cost aware rate optimal router, 2025. URL https://arxiv.org/abs/2502.03261.

Kv Aditya Srivatsa, Kaushal Maurya, and Ekaterina Kochmar. Harnessing the power of multiple minds: Lessons learned from LLM routing. In Shabnam Tafreshi, Arjun Akula, João Sedoc, Aleksandr Drozd, Anna Rogers, and Anna Rumshisky (eds.), *Proceedings of the Fifth Workshop on Insights from Negative Results in NLP*, pp. 124–134, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *CoRR*, abs/1811.03115, 2018. URL http://arxiv.org/abs/1811.03115.

Dimitris Stripelis, Zijian Hu, Jipeng Zhang, Zhaozhuo Xu, Alay Dilipbhai Shah, Han Jin, Yuhang Yao, Salman Avestimehr, and Chaoyang He. TensorOpera router: A multi-model router for efficient LLM inference, 2024. URL https://arxiv.org/abs/2408.12320.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. Spectr: Fast speculative decoding via optimal transport. *Advances in Neural Information Processing Systems*, 36, 2024.

Swabha Swayamdipta, Ankur P. Parikh, and Tom Kwiatkowski. Multi-mention learning for reading comprehension with neural cascades. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HyRnez-RW.

Dharmesh Tailor, Aditya Patra, Rajeev Verma, Putra Manggala, and Eric Nalisnick. Learning to Defer to a Population: A Meta-Learning Approach. In *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics*, 2024.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *23rd International Conference on Pattern Recognition, ICPR 2016, Cancún, Mexico, December 4-8, 2016*, pp. 2464–2469. IEEE, 2016.

Tristan Thrush, Christopher Potts, and Tatsunori Hashimoto. Improving pretraining data using perplexity correlations, 2024. URL https://arxiv.org/abs/2409.05816.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models, 2023.

Vivien Tran-Thien. An optimal lossy variant of speculative decoding, 2023. URL https://github.com/vivien000/mentored_decodings. Unsupervised Thoughts (Blog). URL: https://github.com/vivien000/mentored_decoding.

Kirill Trapeznikov and Venkatesh Saligrama. Supervised sequential classification under budget constraints. In Carlos M. Carvalho and Pradeep Ravikumar (eds.), *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pp. 581–589, Scottsdale, Arizona, USA, 29 Apr–01 May 2013. PMLR.

Neeraj Varshney and Chitta Baral. Model cascading: Towards jointly improving efficiency and accuracy of NLP systems. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11007–11021, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pp. I–I, 2001. doi: 10.1109/CVPR.2001.990517.

Marija Šakota, Maxime Peyrard, and Robert West. Fly-swat or cannon? Cost-effective language model choice via meta-modeling. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, WSDM '24, pp. 606–615, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703713. doi: 10.1145/3616855.3635825. URL https://doi.org/10.1145/3616855.3635825.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. Efficient large language models: A survey. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=bsCCJHbO8A. Survey Certification.

Congchao Wang, Sean Augenstein, Keith Rush, Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Aditya Krishna Menon, and Alec Go. Cascade-aware training of language models, 2024a. URL https://arxiv.org/abs/2406.00060.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models, 2024b. URL https://arxiv.org/abs/2401.00368.

Xiaofang Wang, Dan Kondratyuk, Eric Christiansen, Kris M. Kitani, Yair Movshovitz-Attias, and Elad Eban. Wisdom of committees: An overlooked approach to faster and more accurate models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=MvO2t0vbs4-.

Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, Fisher Yu, and Joseph E. Gonzalez. IDK cascades: Fast deep learning by learning not to overthink. In Amir Globerson and Ricardo Silva (eds.), *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pp. 580–590. AUAI Press, 2018.

Yiding Wang, Kai Chen, Haisheng Tan, and Kun Guo. Tabi: An efficient multi-level inference system for large language models. In *Proceedings of the Eighteenth European Conference on Computer Systems*, EuroSys '23, pp. 233–248, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394871. doi: 10.1145/3552326.3587438. URL https://doi.org/10.1145/3552326.3587438.

Herbert Woisetschläger, Ryan Zhang, Shiqiang Wang, and Hans-Arno Jacobsen. Dynamically learned test-time model routing in language model zoos with service level guarantees, 2025. URL https://arxiv.org/abs/2505.19947.

Ji Xin, Rodrigo Nogueira, Yaoliang Yu, and Jimmy Lin. Early exiting BERT for efficient document ranking. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pp. 83–88, Online, November 2020. Association for Computational Linguistics.

Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. Does unsupervised architecture representation learning help neural architecture search? In *NeurIPS*, 2020.

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. Large language model cascades with mixture of thought representations for cost-efficient reasoning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=6okaSfANzh.

Zesen Zhao, Shuowei Jin, and Z Morley Mao. Eagle: Efficient training-free router for multi-llm inference. *arXiv preprint arXiv:2409.15518*, 2024a.

Ziyu Zhao, Leilei Gan, Guoyin Wang, Wangchunshu Zhou, Hongxia Yang, Kun Kuang, and Fei Wu. LoraRetriever: Input-aware LoRA retrieval and composition for mixed tasks in the wild. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4447–4462, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.263. URL https://aclanthology.org/2024.findings-acl.263/.

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. URL https://openreview.net/forum?id=uccHPGDlao.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. BERT loses patience: Fast and robust inference with early exit. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 18330–18341. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/d4dd111a4fd973394238aca5c05bebe3-Paper.pdf.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Y Zhao, Andrew M. Dai, Zhifeng Chen, Quoc V Le, and James Laudon. Mixture-of-experts with expert choice routing. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=jdJo1HIVinI.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving speculative decoding via knowledge distillation. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=rsY6J3ZaTF.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. A survey on efficient inference for large language models, 2024b. URL https://arxiv.org/abs/2404.14294.

Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. EmbedLLM: Learning compact representations of large language models, 2024. URL https://arxiv.org/abs/2410.02223.

Universal Model Routing for Efficient LLM Inference

Supplementary Material

Table of Contents

A SOCIETAL IMPACT

Our primary contribution is a mathematical formalism for routing amongst multiple dynamic LLMs, with concrete instantiations based on prompt clustering. These are coupled with empirical results on public benchmarks, typically involving LLM accuracy as the primary metric.

We do not foresee immediate negative societal impact from our mathematical framework. While well-studied in the literature, the underlying routing problem itself *could* lend itself to some undesirable outcomes; e.g., a router may overly favour models that produce outputs that are systematically biased on certain data sub-groups. We believe that our framework is amenable to such constraints, and believe that accounting for these is a worthy subject of future research.

B STATIC VS DYNAMIC ROUTING

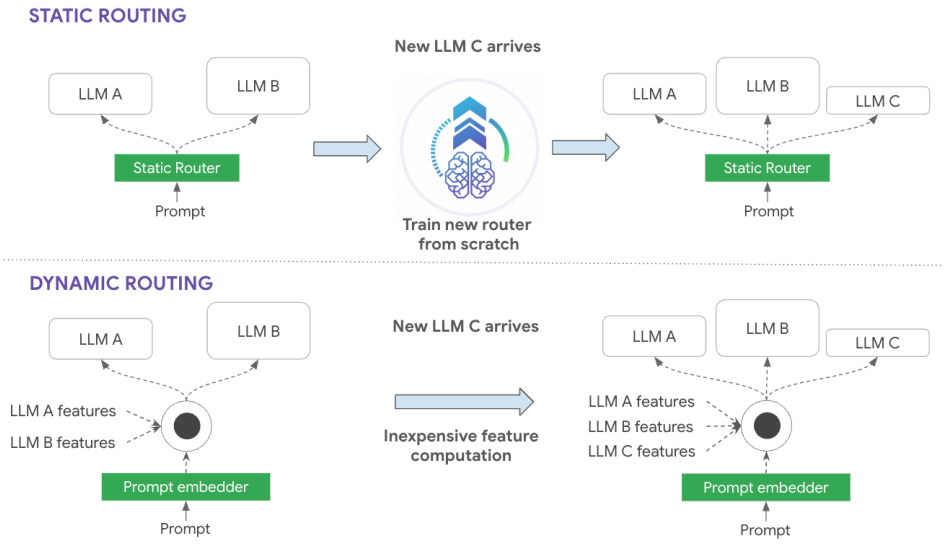


Figure 4: An illustration of static routing and dynamic routing.

Static routing Static routing refers to the setting where the pool of LLMs that a router is trained on is the same one that the router will see at deployment time. That is, the pool is *fixed*, and there is no need for the router to account for new LLMs that may be available after it is trained. If new LLMs are to be part of the LLM pool, then in general the router has to be retrained from scratch. Existing works largely focus on the static routing setting.

Dynamic routing In dynamic routing, one seeks a router that can adapt to a changing LLM pool at deployment time. In particular, a dynamic router is able to add new LLMs to the LLM pool without retraining. *Dynamic routing is the focus of our work.* As formally described in Section 3, our proposed `UniRoute` framework relies on representing each LLM as a feature vector, derived based on predictions on a set of representative prompts.

An illustration to contrast these two settings is provided in Figure 4.

C PROOFS OF RESULTS IN MAIN BODY

In what follows, we use $r_{\mathcal{H}}(\boldsymbol{x})$ and $r(\boldsymbol{x}, \mathcal{H})$ interchangeably. As a shorthand, we define

$$S(r) \doteq \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y},\mathcal{H})}\left[\sum_{m\in[|\mathcal{H}|]} \mathbf{1}(r(\boldsymbol{x},\mathcal{H})=m)\cdot \ell(\boldsymbol{x},\boldsymbol{y},h^{(m)})\right],$$

$$C(r) \doteq \mathbb{E}_{(\boldsymbol{x},\mathcal{H})}\left[\sum_{m\in[|\mathcal{H}|]} \mathbf{1}(r(\boldsymbol{x},\mathcal{H})=m)\cdot c(h^{(m)})\right].$$

The constrained optimization problem in (5) is equivalent to

$$\min_{r\in\mathcal{R}} S(r) \text{ subject to } C(r)\leq B. \tag{14}$$

It will be useful to consider the following related unconstrained optimization objective:

$$L(r,\lambda) = S(r) + \lambda\cdot C(r). \tag{15}$$

For any $\lambda \geq 0$, let r^*_λ be the minimiser of $L(r,\lambda)$.

C.1 INTERMEDIATE RESULTS

We first provide results that will be useful for providing the main claims in §C.2 and §C.3.

Lemma 3. *Given $\lambda \geq 0$, the optimal solution $r^*_\lambda \in \arg\min_r L(r,\lambda)$ to the unconstrained problem in (15) is*

$$r^*_\lambda(\boldsymbol{x},\mathcal{H}) \in \text{argmin}_{m\in[|\mathcal{H}|]}\, \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x},\boldsymbol{y},h^{(m)})\right] + \lambda\cdot c(h^{(m)}).$$

Proof. Starting with the definition of L,

$$\begin{aligned}
L(r,\lambda) &= S(r) + \lambda\cdot C(r)\\
&= \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y},\mathcal{H})}\left[\sum_{m\in[|\mathcal{H}|]} \mathbf{1}(r(\boldsymbol{x},\mathcal{H})=m)\cdot \ell(\boldsymbol{x},\boldsymbol{y},h^{(m)})\right] + \lambda\cdot\mathbb{E}_{(\boldsymbol{x},\mathcal{H})}\left[\sum_{m\in[|\mathcal{H}|]} \mathbf{1}(r(\boldsymbol{x},\mathcal{H})=m)\cdot c(h^{(m)})\right]\\
&\overset{(a)}{=} \mathbb{E}_{\mathcal{H}}\mathbb{E}_{\boldsymbol{x}}\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\sum_{m\in[|\mathcal{H}|]} \mathbf{1}(r(\boldsymbol{x},\mathcal{H})=m)\cdot\left\{\ell(\boldsymbol{x},\boldsymbol{y},h^{(m)}) + \lambda\cdot c(h^{(m)})\right\}\,\middle|\,\boldsymbol{x}\right]\\
&= \mathbb{E}_{\mathcal{H}}\mathbb{E}_{\boldsymbol{x}}\Bigg[\underbrace{\sum_{m\in[|\mathcal{H}|]} \mathbf{1}(r(\boldsymbol{x},\mathcal{H})=m)\cdot\left\{\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x},\boldsymbol{y},h^{(m)})\middle|\,\boldsymbol{x}\right] + \lambda\cdot c(h^{(m)})\right\}}_{\mathcal{L}_{\mathcal{H},\boldsymbol{x}}}\Bigg],
\end{aligned}$$

where (a) uses the fact that the draw of $\mathcal{H}$ is independent of the draw of $(\boldsymbol{x},\boldsymbol{y})$. The last line makes it clear that for any fixed $\mathcal{H}\sim\mathfrak{H}$, and any fixed $\boldsymbol{x}$, to minimize the overall loss, the router ought to route to the model that has the lowest cost-adjusted loss $\mathcal{L}_{\mathcal{H},\boldsymbol{x}}$. Thus,

$$r^*(\boldsymbol{x},\mathcal{H}) \in \text{argmin}_{m\in[|\mathcal{H}|]}\, \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x},\boldsymbol{y},h^{(m)})\right] + \lambda\cdot c(h^{(m)}).$$

□

Lemma 4. *For any $\lambda \geq 0$, let r^*_λ be the minimiser of $L(r,\lambda)$. Then, for any r, if $C(r^*_\lambda) \geq C(r)$, then $S(r^*_\lambda) \leq S(r)$.*

Proof. Since r^*_λ minimises $L(r,\lambda)$, for any r, we have $L(r^*_\lambda,\lambda) \leq L(r,\lambda)$. That is,

$$\begin{aligned}
&S(r^*_\lambda) + \lambda\cdot C(r^*_\lambda) \leq S(r) + \lambda\cdot C(r)\\
\iff &S(r^*_\lambda) \leq S(r) + \lambda\cdot\left(C(r) - C(r^*_\lambda)\right).
\end{aligned} \tag{16}$$

Since $\lambda\geq 0$, it follows that $S(r^*_\lambda)\leq S(r)$ by the assumption that $C(r) - C(r^*_\lambda)\leq 0$. □

Lemma 5. *Let r^*_λ be the minimiser of $L(r, \lambda)$. For $\lambda, \lambda' \geq 0$, $\lambda \geq \lambda'$ if and only if $C(r^*_\lambda) \leq C(r^*_{\lambda'})$. In other words, $\lambda \mapsto C(r^*_\lambda)$ is a non-increasing function. Hence, $\sup_{\lambda \geq 0} C(r^*_\lambda) = C(r^*_0)$.*

Proof. Since $r^*_\lambda \in \arg\min_r L(r, \lambda)$, by definition, we have

$$S(r^*_\lambda) + \lambda \cdot C(r^*_\lambda) \leq S(r) + \lambda \cdot C(r),$$

for any r, including $r^*_{\lambda'}$. This means

$$S(r^*_\lambda) + \lambda \cdot C(r^*_\lambda) \leq S(r^*_{\lambda'}) + \lambda \cdot C(r^*_{\lambda'}).$$

In a symmetric manner,

$$S(r^*_{\lambda'}) + \lambda' \cdot C(r^*_{\lambda'}) \leq S(r^*_\lambda) + \lambda' \cdot C(r^*_\lambda).$$

Adding the above two inequalities, we have

$$\begin{aligned}
&S(r^*_\lambda) + \lambda \cdot C(r^*_\lambda) + S(r^*_{\lambda'}) + \lambda' \cdot C(r^*_{\lambda'}) \leq S(r^*_{\lambda'}) + \lambda \cdot C(r^*_{\lambda'}) + S(r^*_\lambda) + \lambda' \cdot C(r^*_\lambda) \\
&\qquad \implies \lambda \cdot C(r^*_\lambda) + \lambda' \cdot C(r^*_{\lambda'}) \leq \lambda \cdot C(r^*_{\lambda'}) + \lambda' \cdot C(r^*_\lambda) \\
\implies &\lambda \cdot (C(r^*_\lambda) - C(r^*_{\lambda'})) + \lambda' \cdot (C(r^*_{\lambda'}) - C(r^*_\lambda)) \leq 0 \\
&\qquad \implies (\lambda - \lambda')\,(C(r^*_\lambda) - C(r^*_{\lambda'})) \leq 0.
\end{aligned}$$

The last inequality above implies that $\lambda \geq \lambda'$ if and only if $C(r^*_\lambda) \leq C(r^*_{\lambda'})$. □

Lemma 6. *Let $\lambda, \lambda' \geq 0$ such that $\lambda \leq \lambda'$. Given $\boldsymbol{x} \in \mathcal{X}$ and $\mathcal{H} \in \mathbb{H}$, if $r^*_\lambda(\boldsymbol{x}, \mathcal{H}) \neq r^*_{\lambda'}(\boldsymbol{x}, \mathcal{H})$, then there exists a model pair $h^{(m)}, h^{(k)} \in \mathcal{H}$ with $h^{(m)} \neq h^{(k)}$ such that*

$$\lambda \leq \frac{\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x}, \boldsymbol{y}, h^{(k)})\right] - \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x}, \boldsymbol{y}, h^{(m)})\right]}{c(h^{(m)}) - c(h^{(k)})} \leq \lambda'.$$

Proof. Given $\mathcal{H}$, define $A_m(\lambda, \boldsymbol{x}) \doteq \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x}, \boldsymbol{y}, h^{(m)})\right] + \lambda \cdot c(h^{(m)})$. Recall that

$$r^*_\lambda(\boldsymbol{x}, \mathcal{H}) = \operatorname{argmin}_{m \in [|\mathcal{H}|]} A_m(\lambda, \boldsymbol{x}).$$

Observe that given $(\boldsymbol{x}, \mathcal{H})$, for each $m \in [|\mathcal{H}|]$, $A_m(\lambda, \boldsymbol{x})$ is a one-dimensional affine function of λ. So, $r^*_\lambda(\boldsymbol{x}, \mathcal{H})$ is the index of the affine function that gives the lowest value when evaluated at λ. Fix $\boldsymbol{x}$. Since $\lambda \mapsto A_m(\lambda, \boldsymbol{x})$ is continuous, for any m, if $r^*_\lambda(\boldsymbol{x}, \mathcal{H}) \neq r^*_{\lambda'}(\boldsymbol{x}, \mathcal{H})$, it must mean that the index of the lowest affine function changes as we move from λ to λ'. This implies that there is an index pair m and k (with $m \neq k$) for which $\lambda \mapsto A_m(\lambda, \boldsymbol{x})$ and $\lambda \mapsto A_k(\lambda, \boldsymbol{x})$ cross, as we move from λ to λ'. The cross point is precisely at $\bar{\lambda}$ such that

$$\begin{aligned}
&A_m(\bar{\lambda}, \boldsymbol{x}) = A_k(\bar{\lambda}, \boldsymbol{x}) \\
\iff &\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x}, \boldsymbol{y}, h^{(m)})\right] + \bar{\lambda} \cdot c(h^{(m)}) = \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x}, \boldsymbol{y}, h^{(k)})\right] + \bar{\lambda} \cdot c(h^{(k)}) \\
\iff &\bar{\lambda} = \frac{\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x}, \boldsymbol{y}, h^{(k)})\right] - \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x}, \boldsymbol{y}, h^{(m)})\right]}{c(h^{(m)}) - c(h^{(k)})}.
\end{aligned}$$

□

Lemma 7. *Assume $\boldsymbol{x}$ is a continuous random vector. Then, $\lambda \mapsto C(r^*_\lambda)$ is continuous.*

Proof. We will show that for any $\Delta\lambda \in \mathbb{R}$, $\lim_{\Delta\lambda \to 0} |C(r^*_{\lambda+\Delta\lambda}) - C(r^*_\lambda)| = 0$. Observe that

$$\begin{aligned}
|C(r^*_{\lambda+\Delta\lambda}) - C(r^*_\lambda)| &= \left|\mathbb{E}_{(\boldsymbol{x},\mathcal{H})}\left[c\left(h^{\left(r^*_{\lambda+\Delta\lambda}(\boldsymbol{x},\mathcal{H})\right)}\right)\right] - \mathbb{E}_{(\boldsymbol{x},\mathcal{H})}\left[c\left(h^{\left(r^*_\lambda(\boldsymbol{x},\mathcal{H})\right)}\right)\right]\right| \\
&\overset{(a)}{\leq} \mathbb{E}_{(\boldsymbol{x},\mathcal{H})}\left|c\left(h^{\left(r^*_{\lambda+\Delta\lambda}(\boldsymbol{x},\mathcal{H})\right)}\right) - c\left(h^{\left(r^*_\lambda(\boldsymbol{x},\mathcal{H})\right)}\right)\right| \\
&= \mathbb{E}_{\mathcal{H}} \int_{x \in \mathcal{X}} \left|c\left(h^{\left(r^*_{\lambda+\Delta\lambda}(\boldsymbol{x},\mathcal{H})\right)}\right) - c\left(h^{\left(r^*_\lambda(\boldsymbol{x},\mathcal{H})\right)}\right)\right| p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \doteq \spadesuit,
\end{aligned}$$

where we applied Jensen's inequality at (a). Define $\mathcal{S}_{\Delta\lambda}$ to be the set of input points for which r^*_λ and $r^*_{\lambda+\Delta\lambda}$ make different decisions. Precisely,

$$\mathcal{S}_{\Delta\lambda}(\mathcal{H}) \doteq \left\{\boldsymbol{x} \mid r^*_{\lambda+\Delta\lambda}(\boldsymbol{x},\mathcal{H}) \neq r^*_\lambda(\boldsymbol{x},\mathcal{H})\right\}.$$

Let $c_{\max} \doteq \max_{h\in\mathcal{H}_{\text{all}}} c(h)$ i.e., the maximum per-query cost of any LLM in our finite universe. The quantity ♠ can be bounded further with

$$\begin{aligned}
\spadesuit &= \mathbb{E}_{\mathcal{H}} \int_{x\in\mathcal{S}_{\Delta\lambda}(\mathcal{H})} \left| c\left(h^{\left(r^*_{\lambda+\Delta\lambda}(\boldsymbol{x},\mathcal{H})\right)}\right) - c\left(h^{\left(r^*_{\lambda}(\boldsymbol{x},\mathcal{H})\right)}\right)\right| p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \\
&\leq 2c_{\max}\mathbb{E}_{\mathcal{H}} \int_{x\in\mathcal{S}_{\Delta\lambda}(\mathcal{H})} p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \\
&= 2c_{\max}\mathbb{E}_{\mathcal{H}}\mathbb{P}(\mathcal{S}_{\Delta\lambda}(\mathcal{H})).
\end{aligned}$$

The proof now amounts to showing that $\lim_{\Delta\lambda\to 0}\mathbb{E}_{\mathcal{H}}\mathbb{P}(\mathcal{S}_{\Delta\lambda}(\mathcal{H})) = 0$.

Define a shorthand $\lambda_{m,k}(\boldsymbol{x},\mathcal{H}) \doteq \frac{\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x},\boldsymbol{y},h^{(k)})\right] - \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x},\boldsymbol{y},h^{(m)})\right]}{c(h^{(m)}) - c(h^{(k)})}$ where $h^{(m)}, h^{(k)} \in \mathcal{H}$. We start by expanding $\mathcal{S}_{\Delta\lambda}(\mathcal{H})$ to have

$$\begin{aligned}
\mathcal{S}_{\Delta\lambda}(\mathcal{H}) &= \left\{\boldsymbol{x} \mid r^*_{\lambda+\Delta\lambda}(\boldsymbol{x},\mathcal{H}) \neq r^*_\lambda(\boldsymbol{x},\mathcal{H})\right\} \\
&\overset{(a)}{\subseteq} \left\{\boldsymbol{x} \mid \exists k \neq m :\ \lambda_{m,k}(\boldsymbol{x},\mathcal{H}) \in [\lambda, \lambda+\Delta\lambda]\right\} \\
&= \bigcup_{m\neq k} \left\{\boldsymbol{x} \mid \lambda_{m,k}(\boldsymbol{x},\mathcal{H}) \in [\lambda, \lambda+\Delta\lambda]\right\} \\
&\doteq \mathcal{F}_{\Delta\lambda}(\mathcal{H}),
\end{aligned}$$

where at (a) the subset relationship is due to Lemma 6. Since $\mathcal{S}_{\Delta\lambda}(\mathcal{H}) \subseteq \mathcal{F}_{\Delta\lambda}(\mathcal{H})$, we have

$$\begin{aligned}
\spadesuit &\leq 2c_{\max}\mathbb{E}_{\mathcal{H}}\mathbb{P}(\mathcal{F}_{\Delta\lambda}(\mathcal{H})) \\
&\overset{(a)}{\leq} 2c_{\max}\mathbb{E}_{\mathcal{H}} \sum_{m\neq k} \mathbb{P}\left(\left\{\boldsymbol{x} \mid \lambda_{m,k}(\boldsymbol{x},\mathcal{H}) \in [\lambda, \lambda+\Delta\lambda]\right\}\right) \\
&= 2c_{\max}\mathbb{E}_{\mathcal{H}} \sum_{m\neq k} \mathbb{P}\left(\lambda_{m,k}(\boldsymbol{x},\mathcal{H}) \in [\lambda, \lambda+\Delta\lambda]\right),
\end{aligned}$$

where at (a) we use the union bound. Note that there are a finite number of summands because $\mathcal{H}_{\text{all}}$ is finite. Since $\boldsymbol{x}$ is a continuous random variable, for any $\mathcal{H}$, $\lambda_{m,k}(\boldsymbol{x},\mathcal{H})$ is also a continuous random variable. As $\Delta\lambda \to 0$, we have $\mathbb{P}\left(\lambda_{m,k}(\boldsymbol{x},\mathcal{H}) \in [\lambda, \lambda+\Delta\lambda]\right) \to 0$ for any m, k. This means ♠ $\to 0$ and thus $\lim_{\Delta\lambda\to 0} |C(r^*_{\lambda+\Delta\lambda}) - C(r^*_\lambda)| = 0$. □

Proposition 8. *Suppose $\mathbb{P}(\boldsymbol{y}|\boldsymbol{x})$ and $\mathbb{P}(\boldsymbol{x})$ are continuous in $\boldsymbol{x}$. Let $r^*_0 \in \arg\min_r L(r,0)$ where $L(r,\lambda) = S(r) + \lambda\cdot C(r)$ (see also (15)). For any budget $B \in (0, C(r^*_0)]$ in the constrained problem in (5) (or equivalently in (14)), there exists $\lambda_{\mathfrak{H}} \geq 0$ such that the minimiser of $L(r,\lambda_{\mathfrak{H}})$ also minimises (5).*

Proof. Since $\mathbb{P}(\boldsymbol{y}|\boldsymbol{x})$ and $\mathbb{P}(\boldsymbol{x})$ are continuous in $\boldsymbol{x}$, and $\lambda \mapsto C(r^*_\lambda)$ is continuous (by Lemma 7), there exists $\lambda_B \geq 0$ such that $C(r^*_{\lambda_B}) = B$. Applying Lemma 4 with this choice of λ_B implies that

$$S(r^*_{\lambda_B}) \leq S(r),$$

for any r such that $C(r) \leq C(r^*_{\lambda_B}) = B$. This proves that $r^*_{\lambda_B}$ is the minimiser of (5). Setting $\lambda_{\mathfrak{H}} = \lambda_B$ completes the proof. □

C.2 Proof of Proposition 1

Proposition (Restated)**.** *Assume that $\boldsymbol{x} \sim \mathbb{P}$ and $\eta(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{y} \mid \boldsymbol{x})$ are continuous random variables. Then, for any LLM meta-distribution $\mathfrak{H}$, and budget $B \in (0, C(r^*_0)]$ (see §C for the definitions of C and r_0), there exists $\lambda_{\mathfrak{H}} \geq 0$ such that the optimal dynamic router r^* for the constrained optimization in (5) is*

$$r^*(\boldsymbol{x},\mathcal{H}) = \operatorname*{argmin}_{m\in[|\mathcal{H}|]} \left[\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\ell(\boldsymbol{x},\boldsymbol{y},h^{(m)})\right] + \lambda_{\mathfrak{H}}\cdot c(h^{(m)})\right].$$

Note that the continuity assumption on the data distribution applies to the setting where $\boldsymbol{x}$ can be represented via a fixed-length sentence embedding (i.e., a Euclidean vector). This is the primary setting of this work.

Proof. Under the continuity assumption on $\mathbb{P}$, by Proposition 8, there exists $\lambda_{\mathfrak{H}}) \geq 0$ such that the minimiser of the unconstrained objective $L(r, \lambda_{\mathfrak{H}})$ (see (15)) also minimises (5). By Lemma 3, r^* is established.

□

C.3 Proof of Proposition 2

Proposition (Restated)**.** *For a set of LLMs $\mathcal{H}$, let r^* denote the Bayes-optimal routing rule in Proposition 1. For any $\boldsymbol{x} \in \mathcal{X}$ and $h^{(m)} \in \mathcal{H}$, let:*

$$\Delta_k(\boldsymbol{x}, h^{(m)}) = \left| \mathbb{P}_{\boldsymbol{y}|\boldsymbol{x}, z=k} \left[h(\boldsymbol{x}) \neq \boldsymbol{y} \right] - \Psi_k^*(h^{(m)}) \right|.$$

Let $R_{01}(r, \mathcal{H}_{\text{te}}) \doteq \sum_n \mathbb{P}\left[h_{\text{te}}^{(n)}(\boldsymbol{x}) \neq \boldsymbol{y} \wedge r(\boldsymbol{x}, \mathcal{H}_{\text{te}}) = n \right]$ denote the 0-1 risk. Then under the regularity condition on $\mathbb{P}$ in Proposition 1, the difference in 0-1 risk between $\tilde{r}^*$ and r^* can be bounded as:

$$\mathbb{E}_{\mathcal{H}_{\text{te}}} \left[R_{01}(\tilde{r}^*, \mathcal{H}_{\text{te}}) \right] \leq \mathbb{E}_{\mathcal{H}_{\text{te}}} \left[R_{01}(r^*, \mathcal{H}_{\text{te}}) \right] + \mathbb{E}_{(\boldsymbol{x}, \mathcal{H}_{\text{te}})} \left[\max_{m \in [|\mathcal{H}_{\text{te}}|], k \in [K]} \Delta_k(\boldsymbol{x}, h_{\text{te}}^{(n)}) \right].$$

Remark C.1. The bound in Proposition 2 provides us with intuition for when `UniRoute` (K-means) is effective in practice. Specifically, it suggests that the quality gap between cluster-based routing and the optimal rule in (7) is bounded by the gap between the LLM's error on a prompt versus the average error on its constituent cluster. So when this gap is small, we expect cluster-based routing to provide a good quality-cost trade-off. This happens when the input queries form clear clusters, and is the case, for instance, in EmbedLLM which aggregates multiple benchmark datasets. As shown in Figure 2, `UniRoute` (K-Means) is highly performant on this dataset. By contrast, on Headlines, it is not as effective because all prompts are about news related to finance, and the cluster structure is less clear.

We now start with a preliminary for proving Proposition 2. For simplicity, we will refer to $\mathcal{H}_{\text{te}}$ simply by $\mathcal{H}$, and $h_{\text{te}}^{(m)}$ by $h^{(m)}$. We define a proxy risk objective:

$$\tilde{R}_{01}(r, \mathcal{H}) = \mathbb{E}_{\boldsymbol{x}, z} \left[\sum_{m \in [|\mathcal{H}|]} \Psi_z^*(h^{(m)}) \cdot \mathbf{1}\left[r(\boldsymbol{x}, \mathcal{H}) = m \right] \right], \tag{17}$$

where the expectation is over the joint distribution over $(\boldsymbol{x}, z)$ (and not $(\boldsymbol{x}, \boldsymbol{y})$).

Consider solving a variant of the constrained optimization problem in (5) where the original risk objective is replaced with the proxy risk in (17):

$$\begin{aligned} &\min_r \ \mathbb{E}_{\mathcal{H}} \left[\tilde{R}_{01}(r, \mathcal{H}) \right] \\ &\text{s.t.} \quad \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}, \mathcal{H})} \left[\sum_{m \in [|\mathcal{H}|]} c^{(m)} \cdot \mathbf{1}[r(\boldsymbol{x}, \mathcal{H}) = m] \right] \leq B. \end{aligned} \tag{18}$$

We can then show that the optimal solution to above proxy constrained optimization problem admits the same form as the cluster-based routing rule $\tilde{r}^*$ in (13):

Lemma C.2. *Under the assumption on $\mathbb{P}$ in Proposition 2, for any set of models $\mathcal{H}$, the minimizer of the proxy constrained optimization problem in (18) is given by:*

$$\tilde{r}^*(\boldsymbol{x}, \mathcal{H}) = \operatorname*{argmin}_{m \in [|\mathcal{H}|]} \sum_{k \in [K]} \mathbb{P}(z = k | \boldsymbol{x}) \cdot \Psi_k^*(h^{(m)}) + \lambda \cdot c^{(m)},$$

for some $\lambda \geq 0$.

We will also find it useful to bound the difference between the original risk $R_{01}(r, \mathcal{H})$ and the proxy risk in (17):

Lemma C.3. *For any routing rule r and fixed $\mathcal{H}$,*

$$\left| R_{01}(r, \mathcal{H}) - \tilde{R}_{01}(r, \mathcal{H}) \right| \leq \mathbb{E}_{\boldsymbol{x}} \left[\max_{m \in [|\mathcal{H}|], k \in [K]} \Delta_k(\boldsymbol{x}, h^{(m)}) \right].$$

We are now ready to prove Proposition 2:

Proof. The excess risk we wish to bound is given by:

$$\begin{aligned}
&\mathbb{E}_{\mathcal{H}} \left[R_{01}(\tilde{r}^*, \mathcal{H}) - R_{01}(r^*, \mathcal{H}) \right] \\
&= \mathbb{E}_{\mathcal{H}} \left[\left\{ R_{01}(\tilde{r}^*, \mathcal{H}) - \tilde{R}_{01}(\tilde{r}^*, \mathcal{H}) \right\} + \tilde{R}_{01}(\tilde{r}^*, \mathcal{H}) - \tilde{R}_{01}(r^*, \mathcal{H}) + \left\{ \tilde{R}_{01}(r^*, \mathcal{H}) - R_{01}(r^*, \mathcal{H}) \right\} \right] \\
&\overset{(a)}{\leq} \mathbb{E}_{\mathcal{H}} \left[\tilde{R}_{01}(\tilde{r}^*, \mathcal{H}) \right] - \mathbb{E}_{\mathcal{H}} \left[\tilde{R}_{01}(r^*, \mathcal{H}) \right] + 2 \cdot \mathbb{E}_{(\boldsymbol{x}, \mathcal{H})} \left[\max_{m \in [|\mathcal{H}|], k \in [K]} \Delta_k(\boldsymbol{x}, h^{(m)}) \right] \\
&\overset{(b)}{\leq} 0 + 2 \cdot \mathbb{E}_{(\boldsymbol{x}, \mathcal{H})} \left[\max_{m \in [|\mathcal{H}|], k \in [K]} \Delta_k(\boldsymbol{x}, h^{(m)}) \right],
\end{aligned}$$

as desired. To derive (a), we apply Lemma C.3 to bound the first and third terms. To derive (b), we use the fact that $\tilde{r}^*$ is the minimizer of the proxy-risk $\mathbb{E}_{\mathcal{H}} \left[\tilde{R}_{01}(\cdot, \mathcal{H}) \right]$ subject to the budget constraint in (18); since r^* also satisfies the same budget constraint, it has an equal or higher expected risk than $\tilde{r}^*$. □

We now prove Lemmas C.2 and C.3.

Proof of Lemma C.2. Under the assumption on $\mathbb{P}$, the constrained problem in (18) is equivalent to minimizing the following Lagrangian objective for some Lagrange multiplier λ (Neyman & Pearson, 1933):

$$\begin{aligned}
\mathcal{L} &= \mathbb{E}_{(\boldsymbol{x}, z, \mathcal{H})} \left[\sum_{m \in [|\mathcal{H}|]} \Psi_z^*(h^{(m)}) \cdot \mathbf{1} \left[r(\boldsymbol{x}, \mathcal{H}) = m \right] \right] + \lambda \cdot \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}, \mathcal{H})} \left[\sum_{m \in [|\mathcal{H}|]} \mathbf{1}(r(\boldsymbol{x}, \mathcal{H}) = m) \cdot c^{(m)} \right] \\
&\overset{(a)}{=} \mathbb{E}_{\mathcal{H}} \mathbb{E}_{\boldsymbol{x}} \mathbb{E}_{z|\boldsymbol{x}} \left[\sum_{m \in [|\mathcal{H}|]} \mathbf{1}(r(\boldsymbol{x}, \mathcal{H}) = m) \cdot \left\{ \Psi_z^*(h^{(m)}) + \lambda \cdot c^{(m)} \right\} \right] \\
&= \mathbb{E}_{\mathcal{H}} \mathbb{E}_{\boldsymbol{x}} \underbrace{\left[\sum_{m \in [|\mathcal{H}|]} \mathbf{1}(r(\boldsymbol{x}, \mathcal{H}) = m) \cdot \left\{ \sum_{k \in [K]} \mathbb{P}(z = k | \boldsymbol{x}) \cdot \Psi_k^*(h^{(m)}) + \lambda \cdot c^{(m)} \right\} \right]}_{\mathcal{L}_{\mathcal{H}, \boldsymbol{x}}},
\end{aligned}$$

where (a) uses the fact that the draw of $\mathcal{H}$ is independent of the draw of $(\boldsymbol{x}, \boldsymbol{y})$. The last line makes it clear that for any fixed $\mathcal{H}$ and any fixed $\boldsymbol{x}$, to minimize the overall loss, the router ought to route to the model that has the lowest cost-adjusted loss $\mathcal{L}_{\mathcal{H}, \boldsymbol{x}}$. Thus,

$$\tilde{r}^*(\boldsymbol{x}, \mathcal{H}) = \operatorname*{argmin}_{m \in [|\mathcal{H}|]} \sum_{k \in [K]} \mathbb{P}(z = k | \boldsymbol{x}) \cdot \Psi_k^*(h^{(m)}) + \lambda \cdot c^{(m)}.$$

□

Proof of Lemma C.3. Expanding the original risk, we have:

$$\begin{aligned}
R_{01}(r, \mathcal{H}) &= \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y})} \left[\sum_{m \in [|\mathcal{H}|]} \mathbf{1} \left[h^{(m)}(\boldsymbol{x}) \neq \boldsymbol{y} \right] \cdot \mathbf{1} \left[r(\boldsymbol{x}, \mathcal{H}) = m \right] \right] \\
&= \mathbb{E}_{\boldsymbol{x}} \left[\sum_{m \in [|\mathcal{H}|]} \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}} \left[\mathbf{1} \left[h^{(m)}(\boldsymbol{x}) \neq \boldsymbol{y} \right] \cdot \mathbf{1} \left[r(\boldsymbol{x}, \mathcal{H}) = m \right] \right] \right]
\end{aligned}$$

$$= \mathbb{E}_{\boldsymbol{x}} \left[\sum_{m \in [|\mathcal{H}|]} \mathbb{P}_{\boldsymbol{y}|\boldsymbol{x}} \left[h^{(m)}(\boldsymbol{x}) \neq \boldsymbol{y} \right] \cdot \mathbf{1} \left[r(\boldsymbol{x}, \mathcal{H}) = m \right] \right]$$

$$= \mathbb{E}_{\boldsymbol{x}} \left[\sum_{m \in [|\mathcal{H}|]} \sum_{k \in [K]} \pi_k \cdot \mathbb{P}_{\boldsymbol{y}|\boldsymbol{x}, z=k} \left[h^{(m)}(\boldsymbol{x}) \neq \boldsymbol{y} \right] \cdot \mathbf{1} \left[r(\boldsymbol{x}, \mathcal{H}) = m \right] \right]. \tag{19}$$

Recall that:

$$\begin{aligned} \Delta_k(\boldsymbol{x}, h^{(m)}) &= \left| \mathbb{P}_{\boldsymbol{y} \,|\, \boldsymbol{x}, z=k} \left[h^{(m)}(\boldsymbol{x}) \neq \boldsymbol{y} \right] \; - \; \Psi_k^*(h^{(m)}) \right| \\ &= \left| \mathbb{P}_{\boldsymbol{y} \,|\, \boldsymbol{x}, z=k} \left[h^{(m)}(\boldsymbol{x}) \neq \boldsymbol{y} \right] \; - \; \mathbb{P}_{\boldsymbol{x}', \boldsymbol{y}' \,|\, z=k} \left[h^{(m)}(\boldsymbol{x}') \neq \boldsymbol{y}' \right] \right|. \end{aligned}$$

We may next bound (19) in terms of $\Delta_m(\boldsymbol{x}, h^{(m)})$:

$$\begin{aligned} & R_{01}(r, \mathcal{H}) \\ & \leq \mathbb{E}_{\boldsymbol{x}} \left[\sum_{m \in [|\mathcal{H}|]} \sum_{k \in [K]} \pi_k \cdot \left(\mathbb{P}_{\boldsymbol{x}', \boldsymbol{y}' \,|\, z=k} \left[h^{(m)}(\boldsymbol{x}') \neq \boldsymbol{y}' \right] + \Delta_k(\boldsymbol{x}, h^{(m)}) \right) \cdot \mathbf{1} \left[r(\boldsymbol{x}, \mathcal{H}) = m \right] \right] \\ & = \mathbb{E}_{\boldsymbol{x}} \left[\sum_{k \in [K]} \pi_k \cdot \sum_{m \in [|\mathcal{H}|]} \mathbb{P}_{\boldsymbol{x}', \boldsymbol{y}' | z=k} \left[h^{(m)}(\boldsymbol{x}') \neq \boldsymbol{y}' \right] \cdot \mathbf{1} \left[r(\boldsymbol{x}, \mathcal{H}) = m \right] \right] \\ & \qquad + \mathbb{E}_{\boldsymbol{x}} \left[\sum_{k \in [K]} \pi_k \cdot \sum_{m \in [|\mathcal{H}|]} \Delta_k(\boldsymbol{x}, h^{(m)}) \cdot \mathbf{1} \left[r(\boldsymbol{x}, \mathcal{H}) = m \right] \right] \\ & \overset{(a)}{\leq} \mathbb{E}_{\boldsymbol{x}} \left[\sum_{k \in [K]} \pi_k \cdot \sum_{m \in [|\mathcal{H}|]} \mathbb{P}_{\boldsymbol{x}', \boldsymbol{y}' | z=k} \left[h^{(m)}(\boldsymbol{x}') \neq \boldsymbol{y}' \right] \cdot \mathbf{1} \left[r(\boldsymbol{x}, \mathcal{H}) = m \right] \right] + \mathbb{E}_{\boldsymbol{x}} \left[\sum_{k \in [K]} \pi_k \cdot \max_{m \in [|\mathcal{H}|]} \Delta_k(\boldsymbol{x}, h^{(m)}) \right] \\ & \overset{(b)}{\leq} \mathbb{E}_{\boldsymbol{x}} \left[\sum_{k \in [K]} \pi_k \cdot \sum_{m \in [|\mathcal{H}|]} \mathbb{P}_{\boldsymbol{x}', \boldsymbol{y}' | z=k} \left[h^{(m)}(\boldsymbol{x}') \neq \boldsymbol{y}' \right] \cdot \mathbf{1} \left[r(\boldsymbol{x}, \mathcal{H}) = m \right] \right] + \mathbb{E}_{\boldsymbol{x}} \left[\max_{m \in [|\mathcal{H}|], k \in [K]} \Delta_k(\boldsymbol{x}, h^{(m)}) \right] \\ & = \mathbb{E}_{(\boldsymbol{x}, z)} \left[\sum_{m \in [|\mathcal{H}|]} \Psi_z^*(h^{(m)}) \cdot \mathbf{1} \left[r(\boldsymbol{x}, \mathcal{H}) = m \right] \right] + \mathbb{E}_{\boldsymbol{x}} \left[\max_{m \in [|\mathcal{H}|], k \in [K]} \Delta_k(\boldsymbol{x}, h^{(m)}) \right] \\ & = \tilde{R}_{01}(r, \mathcal{H}) + \mathbb{E}_{\boldsymbol{x}} \left[\max_{m \in [|\mathcal{H}|], k \in [K]} \Delta_k(\boldsymbol{x}, h^{(m)}) \right]. \end{aligned}$$

where (a) uses the fact that $\sum_m \mathbf{1} \left[r(\boldsymbol{x}, \mathcal{H}) = m \right] = 1$ and (b) follows from the fact that $\sum_k \pi_k = 1$.

One can similarly show that:

$$R_{01}(r, \mathcal{H}) \; \geq \; \tilde{R}_{01}(r, \mathcal{H}) \; - \; \mathbb{E}_{\boldsymbol{x}} \left[\max_{m \in [|\mathcal{H}|], k \in [K]} \Delta_k(\boldsymbol{x}, h^{(m)}) \right],$$

which completes the proof. □

D ZERO ROUTING

An elementary approach to multi-model routing is to identify the points on the non-decreasing convex hull of the set of cost-risk pairs $\{(c^{(m)}, R(h^{(m)}) : m \in [M]\}$, and to route amongst the corresponding LLMs (Hu et al., 2024b). Specifically, given a budget $B \in (c^{(1)}, c^{(M)})$, we may pick the two nearest costs $c^{(m_1)} \leq B < c^{(m_2)}$ from the non-decreasing convex hull, and route a query to LLM $h^{(m_1)}$ with probability $\frac{c^{(m_2)}-B}{c^{(m_2)}-c^{(m_1)}}$ and to $h^{(m_2)}$ with probability $\frac{B-c^{(m_1)}}{c^{(m_2)}-c^{(m_1)}}$.

This approach can be seen as a *random router* that randomizes between two LLMs, where the LLMs and mixing coefficients are chosen to maximize the expected quality on the validation sample, while satisfying the budget constraint. Despite its simplicity (i.e., the routing decision being agnostic to the input), this approach was noted as a strong baseline in Hu et al. (2024b).

D.1 SPECIAL CASE: OPTIMAL CLUSTER ROUTING RULE FOR $K = 1$

When the number of clusters $K = 1$, the routing rule in (8) returns the same LLM for all queries $\boldsymbol{x}$:

$$r(\boldsymbol{x}, \mathcal{H}_{\text{te}}) = \operatorname*{argmin}_{n \in [N]} \left[\Psi(h_{\text{te}}^{(n)}) + \lambda \cdot c(h_{\text{te}}^{(n)})\right], \tag{20}$$

where $\Psi(h_{\text{te}}^{(n)}) \doteq \frac{1}{N_{\text{val}}} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in S_{\text{val}}} \mathbf{1}[h_{\text{te}}^{(n)}(\boldsymbol{x}) \neq \boldsymbol{y}]$. This rule is closely aligned with the Pareto-random router.

Proposition 9. *For any $\lambda \in \mathbb{R}_{\geq 0}$, the routing rule in (20) returns an LLM on the non-decreasing* convex hull *of the set of cost-risk pairs $\{(c(h_{\text{te}}^{(n)}), r_{01}(h_{\text{te}}^{(n)}) : n \in [N]\}$, where $r_{01}(h_{\text{te}}^{(n)}) = \frac{1}{N_{\text{val}}} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in S_{\text{val}}} \mathbf{1}[h_{\text{te}}^{(n)}(\boldsymbol{x}) \neq \boldsymbol{y}]$.*

Proof. Suppose there exists a $\lambda_1 \in \mathbb{R}_{\geq 0}$ and $m_1 \in \operatorname{argmin}_m \Psi(h_{\text{te}}^{(m)}) + \lambda_1 \cdot c^{(m)}$ such that $(c^{(m_1)}, r_{01}(h^{(m_1)}))$ is not on the non-decreasing convex hull. Then there exists $h^{(m_2)} \in \mathcal{H}$ such that either $c^{(m_2)} < c^{(m_1)}$ and $r_{01}(h^{(m_2)}) \leq r_{01}(h^{(m_1)})$, or $c^{(m_2)} \leq c^{(m_1)}$ and $r_{01}(h^{(m_2)}) < r_{01}(h^{(m_1)})$. In either case, $\Psi(h_{\text{te}}^{(m_2)}) + \lambda_1 \cdot c^{(m_2)} = r_{01}(h_{\text{te}}^{(m_2)}) + \lambda_1 \cdot c^{(m_2)} < r_{01}(h_{\text{te}}^{(m_1)}) + \lambda_1 \cdot c^{(m_1)} = \Psi(h_{\text{te}}^{(m_1)}) + \lambda_1 \cdot c^{(m_1)}$, which contradicts the fact that $m_1 \in \operatorname{argmin}_m [\Psi(h_{\text{te}}^{(m)}) + \lambda_1 \cdot c^{(m)}]$. □

E BUDGET CONTROL AND DEFERRAL CURVES

Ideally, one wishes to find a router that exactly satisfies a given cost budget B in (5). Per Proposition 1, following standard Lagrangian theory, this may be reduced to tuning a certain soft penalty coefficient $\lambda_{\mathfrak{H}}$. This raises the question: how exactly does one pick $\lambda_{\mathfrak{H}}$ corresponding to a given B?

We note at the outset that this question is relevant for *any* routing method (not only ours). Indeed, even in the static routing setting, the question of how to tune λ (i.e., the quality-cost trade-off parameter) is independent of the core technical question of how to design a good router r. To separate these concerns – design of a good routing score estimate, versus design of a good trade-off parameter selection – it is standard in the literature to sweep over λ (i.e., report a deferral curve).

This said, for completeness, we include a discussion on how to tune $\lambda_{\mathfrak{H}}$. First, note that $\lambda_{\mathfrak{H}}$ is not determined during the training phase. Instead, we would need to tune it to meet the budget constraint whenever a new set of LLMs are included in the LLM pool. In practice, this can be done using an inexpensive *line search*: find the smallest for which the routing rule satisfies the budget constraint. This line search is a one-off procedure needed to be run when new LLMs arrive; it requires no re-training, gradient updates, or labeled samples.

Second, we highlight that we are able to decouple the training phase from the estimation of thanks to the optimal rule in (8):

$$r^*(x) = \arg\min_m \ (\text{Loss of LLM } h^{(m)} \text{ on } x) + \lambda_{\mathfrak{H}} \cdot \ (\text{cost of calling } h^{(m)}).$$

With this strategy, we may guarantee that the average cost of inference on the test prompts is within the specified budget. More specifically, suppose we have access to a set of validation prompts that are drawn from the same distribution as the test prompts. by standard generalization theory, the average cost of inference on the test set is also guaranteed (up to sampling errors) to be within budget. This guarantee is similar to a classifier trained on the training set generalizing well to new test samples (the analogue of the classifier here is the routing rule with a single tuning parameter).

F EXPERIMENTAL SETUP

We provide more details on the experiments discussed in Section 6.

F.1 SPLITTING DATA AND LLMS

In the experiments, we split the data into three disjoint portions: train, validation, and test. The set of all LLMs available in each dataset is also split into two disjoint sets: training models and testing models. The relationship of data splits and model splits is summarized in the following table.

Data split / Model split	Train	Validation	Test
Training models	✓	✓	✗
Testing models	✗	✓	✓

- **Training set**. The training examples are meant for router training. Only information of the training models (not testing models) is available in this data portion. The only exceptions are the clairvoyant oracle baselines which are allowed access to correctness labels of testing models on training examples. In other words, unlike other baselines, these oracle methods observe all models during training, and are trained on both training and validation portions. These baselines are meant to establish performance achievable if a router has access to all models.
- **Validation set**. The validation examples are meant to be used to represent new LLMs. For instance, for our proposed `UniRoute` (K-Means) approach, the validation set is used to compute per-cluster performance metrics of each testing LLM observed at test time, to represent it as a feature vector. We assume that the validation set is small so as to keep the cost of adding a new LLM low.
- **Test set**. The test examples are only used for evaluating routing methods.

Testing models represent new models that arrive at deployment time, and are not available for training (except to the clairvoyant fixed-pool router baseline). Training models are meant for router training. For instance, our `UniRoute` (K-Means) approach learns to route among the training models, and is tested on the test set to route among the testing models.

F.2 EVALUATION: DEFERRAL CURVE

Routing performance may be assessed via a *deferral curve* (Jitkrittum et al., 2023; Wang et al., 2024a; Hu et al., 2024b) $\mathcal{C} = \{(B, R(h_{\mathrm{RM}}(\cdot, r_B)) \colon B \in [c^{(1)}, c^{(M)}]\}$, tracing the tradeoff between the cost budget B and loss of the resulting routed model. Specifically, one varies the cost budget $B \in [c^{(1)}, c^{(M)}]$; computes a router $r_B(\cdot)$ for this budget; and plots the resulting expected loss $R(h_{\mathrm{RM}}(\cdot, r_B))$. We may also use a quality metric (e.g., accuracy) in place of the loss to capture quality-cost trade-offs.

F.3 IMPLEMENTATION DETAILS OF `UniRoute` (LEARNEDMAP)

In this section, we give details on the architecture and training of our proposed `UniRoute` (LearnedMap).

Architecture Recall from Section 5.2 that the LearnedMap attempts to learn $\boldsymbol{x} \mapsto \boldsymbol{\Phi}_{\mathrm{clust}}(\boldsymbol{x}; \boldsymbol{\theta})$ parameterised by $\boldsymbol{\theta}$. This is a function that maps an input prompt $\boldsymbol{x}$ to a probability vector $\boldsymbol{\Phi}_{\mathrm{clust}}(\boldsymbol{x}; \boldsymbol{\theta}) \in \Delta^K$ where Δ^K denotes the K-dimensional probability simplex. In experiments, $\boldsymbol{\Phi}_{\mathrm{clust}}(\cdot; \boldsymbol{\theta})$ is defined by an MLP with two hidden layers:

$$\boldsymbol{\Phi}_{\mathrm{clust}}(\boldsymbol{x}; \boldsymbol{\theta}) \doteq (\mathrm{Softmax} \circ \mathrm{FC}(K) \circ \mathsf{H}' \circ \mathsf{H} \circ \mathrm{BN} \circ \boldsymbol{\varphi})(\boldsymbol{x}), \tag{21}$$

$$\mathsf{H} \doteq [\mathrm{ReLU} \circ \mathrm{BN} \circ \mathrm{FC}(128)], \tag{22}$$

where:

- $\circ$ denotes function composition,
- H and H' denote the two, separate (i.e., no parameter sharing) hidden layers of the same architecture as described in (22),
- $\mathrm{FC}(z)$ denotes a fully connected layer with $z \in \mathbb{N}$ outputs,
- ReLU denotes the rectified linear unit i.e., $\mathrm{ReLU}(a) \doteq \max(0, a)$,
- $\mathrm{Softmax}$ denotes a softmax layer,
- BN denotes the batch normalization layer, and
- $\boldsymbol{\varphi}$ denotes a frozen text embedding.

Recall from the experiments in Section 6 that we use Gecko Lee et al. (2024b) for the text embedding $\boldsymbol{\varphi}$.

Training We use Keras (Chollet et al., 2015) for implementing (21). In all experiments, `UniRoute` (LearnedMap) is trained for only 5 epochs using Adam as the optimization algorithm. We observe that training for too long can lead to overfitting. Training batch size is set to 64, and the learning rate is 0.005. Since $\boldsymbol{\varphi}$ is frozen, training $\boldsymbol{\Phi}_{\mathrm{clust}}(\cdot; \boldsymbol{\theta})$ amounts to training the MLP with two hidden layers. It is sufficient to use CPUs for training. For one trial, training takes only a few minutes to complete.

F.4 Router Cost

All routing methods rely on a frozen prompt embedding model $\boldsymbol{\varphi}$. As described in Appendix G.3, $\boldsymbol{\varphi}$ is set be the token probability quantiles of the Gemma 2B model in the Headlines dataset. For other datasets, we use Gecko 1B model (Lee et al., 2024b) for $\boldsymbol{\varphi}$. Importantly, all methods rely on the same embedding model, and thus share the same overhead for prompt embedding.

At inference time, for `UniRoute` (K-Means), deciding the LLM to route to for a given test query involves computing the text embedding, and finding the nearest centroid out of K centroids (recall that K-means is run only once on the training set). For `UniRoute` (LearnedMap), after computing the prompt embedding, we pass the embedding through a small MLP as described in (21). Compared to the sizes of candidate LLMs (up to 60+B in EmbedLLM, for instance), invoking the text embedding model and performing a few operations after (centroid lookup, or invoking a small MLP) incur a negligible cost.

When plotting the deferral curves, we do not include the cost of the routing model. Including the router's overhead will simply shift all the deferral curves to the right by a small amount, without changing the relative ordering of the methods.

G Additional Experimental Results

We present additional experimental results we omitted in the main text.

G.1 Hyper-parameter Tuning

To tune K in K-NN, K-means, and the proposed LearnedMap, for each candidate K, we represent the training LLM using correctness labels in the training set, evaluate the routing rule for the training LLMs on the validation set, and measure the area under the deferral curve. This evaluation metric can be seen as the average improvement in quality per unit cost. The parameter with the maximum area is then chosen.

G.2 Hyper-parameter Choices & Statistical Significance

There are three methods that we consider in the experiments in Section 6 that depend on a hyperparameter K. Specifically, K in K-NN refers to the number of nearest neighbors, and K in `UniRoute` (K-means) and `UniRoute` (LearnedMap) refers to the number of clusters. In Figure 2, we report the performance of these methods with the best K found on each dataset separately. We now describe the validation procedure we used to select the best K.

K-NN For each candidate K, and each prompt in the validation set, find the K nearest queries in the training set (in the Gecko embedding space). Route each prompt in the validation set to the most appropriate *training* LLM according to the routing rule (7) where $\gamma^{(m)}$ is estimated with (4). Produce a deferral curve on the validation set, and compute the normalized area under such curve. Select K that maximizes the area. The list of candidate K's is set to be from 5 to one third of the validation sample size.

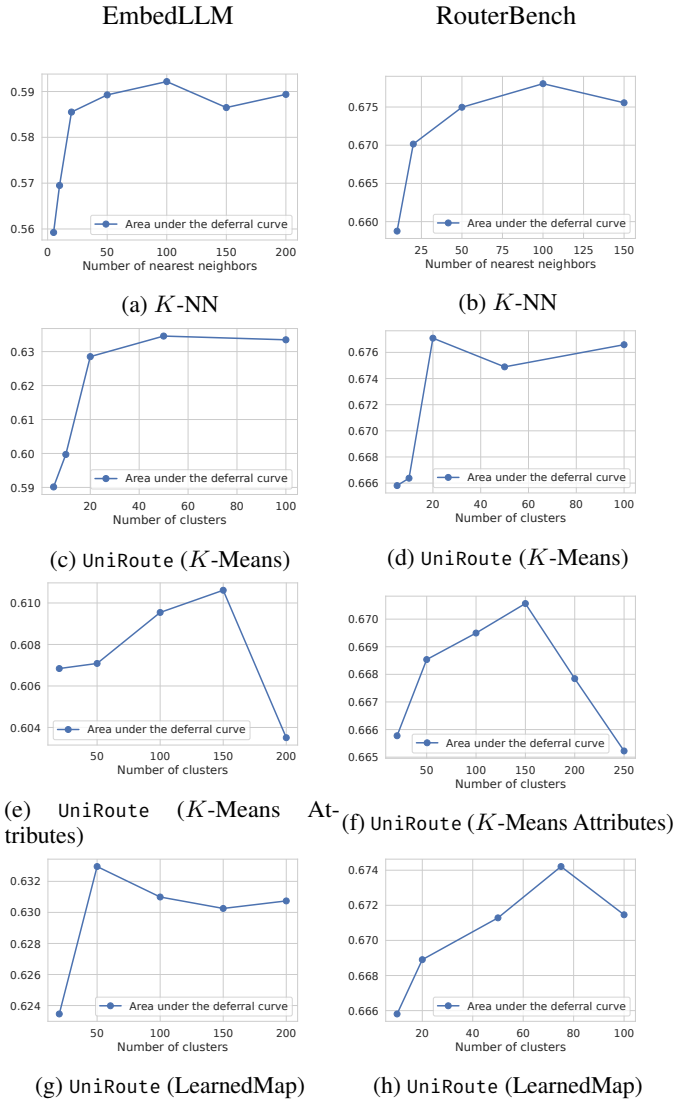
`UniRoute` **(K-Means)** For each candidate K, perform K-means on the training set using Gecko embeddings (Lee et al., 2024b). Compute the feature vector representation of each *training* LLM on the training set using (12). For each prompt in the validation set, find the nearest cluster, and route the prompt to the most appropriate *training* LLM according to the routing rule (8). Produce a deferral curve on the validation set, and compute the normalized area under such curve. Select K that maximizes the area. The list of candidate K's is from 3 to the number of validation sample size, divided by 50.

`UniRoute` **(K-Means Attributes)** An alternate approach that we consider in Appendix G.4 is to construct a binary vector of *prompt attributes*, denoting whether a prompt possesses certain characteristics (Li et al., 2024b;c), e.g., whether it requires multi-step reasoning, seeks a single correct answer, and so on. These can be seen as a generalised "task". Compared to a general purpose text embedding, such a representation is a coarser representation; on the other hand, for the purposes of model routing, this can help mitigate overfitting.

Concretely, we parameterize the prompt embedding model to be $\Phi(\boldsymbol{x}) = \sigma(\mathbf{V}^\top \boldsymbol{\varphi}_{\text{Gecko}}(\boldsymbol{x}))$ where $\mathbf{V} \in \mathbb{R}^{768\times 7}$, and σ denotes the sigmoid function. Train each head $\mathbf{v}_j \in \mathbb{R}^{768}$ (with $\boldsymbol{\varphi}_{\text{Gecko}}$ frozen) by minimizing the sigmoid cross entropy to predict whether the j-th semantic attribute is active on each input prompt. We use the seven prompt difficulty attributes as described in Li et al. (2024b), and prompt Gemini 1.5 Pro 002 to annotate each binary attribute on each training example. Once the prompt embedding model Φ is trained, we freeze it, and perform the same hyperparameter selection procedure as used for K-means (Gecko) by replacing the Gecko embedding function with Φ.

`UniRoute` **(LearnedMap)** For each candidate K, perform K-means on the training set using Gecko embeddings. Compute the feature vector representation of each *training* LLM on the training set using (12). Parameterize the cluster assignment map with a softmax-dense layer as described in Section 5.2, resulting in a parameter $\boldsymbol{\theta}$ to learn. Learn the parameter by minimizing the binary sigmoid cross entropy on the training with the labels given by the correctness labels of the *training* LLMs. With the cluster map trained, for each prompt in the validation set, route the prompt to the most appropriate training LLM according to the routing rule (8). Produce a deferral curve on the validation set, and compute the normalized area under such curve. Select K that maximizes the area. The list of candidate K's is from 3 to the number of validation sample size, divided by 50.

Effect of K Figure 5 shows the area under the deferral curve (on the validation set) vs candidate parameter K. Importantly, the testing models and the test set are never used in the above hyperparameter selection process.



(a) K-NN (b) K-NN

(c) `UniRoute` (K-Means) (d) `UniRoute` (K-Means)

(e) `UniRoute` (K-Means Attributes) (f) `UniRoute` (K-Means Attributes)

(g) `UniRoute` (LearnedMap) (h) `UniRoute` (LearnedMap)

Figure 5: Validation performance of methods considered in Figure 2 and Appendix G.4: K-NN, `UniRoute` (K-Means), `UniRoute` (K-Means Attributes), and `UniRoute` (LearnedMap). See Appendix G.2 for more details.

MLP (Clairvoyant) For this oracle baseline, as per (Hu et al., 2024b), we fix the number of hidden layers to two and the number of hidden nodes in each hidden layer to 100. We fit the MLP on the combined training and validation set to predict a quality score for each test LLM, and route via (2). For training, we ran 10 epochs of Adam optimization with a learning rate of 10^{-3} and batch size 64, and picked the checkpoint that yielded the best quality metric on the validation set.

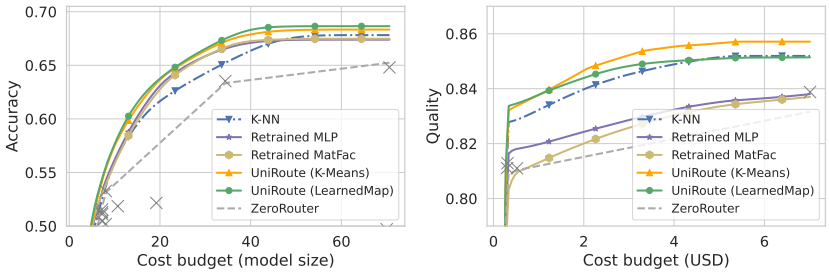
Matrix Factorization (Clairvoyant) For this oracle baseline, we fit a factorized model $\mathbf{A} \cdot \mathbf{B}$, with $\mathbf{A} \in \mathbb{R}^{D_{\mathrm{P}} \times d}, \mathbf{B} \in \mathbb{R}^{d \times m}$, where D_{P} is the dimension of the pretrained query embedding, d is an intermediate dimension, and N is the number of test LLMs to route to. We choose d using the same procedure used to pick the hyper-parameter K above. For a query $\boldsymbol{x} \in \mathcal{X}$ and pre-trained $\boldsymbol{\varphi}(\boldsymbol{x}) \in \mathbb{R}^{D_{\mathrm{P}}}$, this router predicts N LLM scores via $\mathbf{A} \cdot \mathbf{B} \cdot \boldsymbol{\varphi}(\boldsymbol{x})$, and routes via (2). For training,

we ran 10 epochs of Adam with a learning rate of 10^{-3} and batch size 64, and picked the checkpoint that yielded the best quality metric on the validation set.

Statistical Significance For the table in Figure 2 (Top), we repeat the experiments over 400 trials, and report statistical significance (via the sign test) at signficance level $\alpha = 0.01$. For the plots of area vs. validation sample size in Figure 2 (Bottom), we repeat the experiments over 100 trials, and report 96% confidence intervals (i.e. two-sigma standard errors).

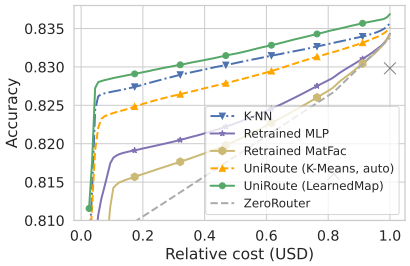
G.3 Deferral Curves and Additional Comparisons

Method \ Dataset	EmbedLLM		SPROUT o3-mini		Headlines		RouterBench		Math+Code	
	QNC ↓	Area ↑	QNC ↓	Area ↑	QNC ↓	Area ↑	QNC ↓	Area ↑	QNC ↓	Area ↑
ZeroRouter (Hu et al., 2024b)	87.5% *	.607 *	100.0% *	.820 *	88.0% *	.819 *	99.9%	.689 *	82.8% *	.395 *
K-NN (Hu et al., 2024b; Shnitzer et al., 2023)	45.9% *	.636 *	29.6% *	.844 *	43.7% *	.830 *	99.7%	.707 *	25.7%	.487 *
Retrained MLP (Hu et al., 2024b)	35.9% *	.641 *	80.9% *	.829 *	74.2% *	.823 *	99.2%	.710 *	—	—
Retrained MatFac (Ong et al., 2025; Zhuang et al., 2024)	36.6% *	.640 *	84.2% *	.825 *	80.9% *	.821 *	99.3%	.708 *	—	—
`UniRoute` (K-Means)	33.7%	.649 *	19.6%	.850	56.9% *	.828 *	99.4%	.712	25.7%	.490
`UniRoute` (LearnedMap)	33.1%	.652	23.4%	.846	34.9%	.832	99.6%	.711	—	—
Clairvoyant MLP	26.9%	.664	4.5%	.859	18.1%	.835	95.2%	.723	25.1%	.500
Clairvoyant MatFac	27.5%	.662	5.0%	.857	13.7%	.835	99.4%	.721	—	—

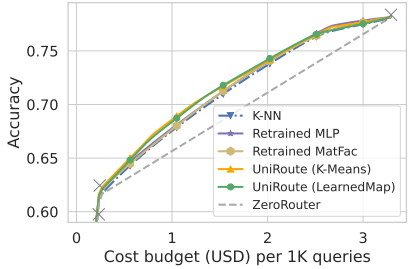


(a) EmbedLLM

(b) SPROUT o3-mini



(c) Headlines



(d) RouterBench

Figure 6: Deferral curves and router evaluation metrics (QNC, and Area) for different methods in the dynamic pool setting. Clairvoyant MLP (Hu et al., 2024a) and Clairvoyant MatFac (Ong et al., 2025; Zhuang et al., 2024) are oracle methods that observe correctness labels of testing LLMs on the large set of training prompts. ZeroRouter (Hu et al., 2024b), K-NN Hu et al. (2024b); Shnitzer et al. (2023), and retraining variants of MLP and matrix factorization are baselines applicable to the dynamic LLM pool setting.

As described in Section 6, we randomly split examples of each dataset into training, validation and testing over 400 independent trials. Figure 6 presents the mean deferral curves of all the methods we consider. The Area and QNC statistics reported in Figure 2 are derived from these curves. All results here are for the dynamic LLM pool setting as considered in Figure 2. In Figure 6, we present two additional oracle methods:

- ***Clairvoyant** MLP and **Clairvoyant** MatFac*: Idealized versions of the MLP and factorised routers, with one output per train and test LLM. These are trained on the combined training and validation set (with correctness labels from both the train and test LLMs), and serve as *oracles* to estimate an *upper bound* on router performance on unseen LLMs (assume the set of LLMs is fully known). These **oracle** baselines provide an estimate of the *performance achievable when all LLMs are observed*. Recall that other non-clairvoyant methods (including `UniRoute`) do not observe test LLMs on the (large) training data split (see Appendix F.1).

Headlines We consider the Headlines dataset (Sinha & Khandait, 2021; Chen et al., 2023b) consisting of 10K prompts in total. Each prompt asks an LLM to determine the price direction (up, down, neutral, or none) of an item in a list of news headlines. There are 12 LLMs of various sizes in this dataset. Each LLM has a distinct cost (USD) of processing one prompt, which is a function of the

input length, output length, and a fixed cost per request (see Table 1 in Chen et al. (2023b)). We hold out six LLMs for training and the other six LLMs for testing:

- Training LLMs: Textsynth FAIRSEQ, OpenAI GPT-4, OpenAI GPT-Curie, Textsynth GPT-Neox, AI21 J1-Large, Cohere Xlarge;
- Test LLMs: OpenAI ChatGPT, OpenAI GPT-3, Textsynth GPT-J, AI21 J1-Grande, AI21 J1-Jumbo, Cohere Medium.

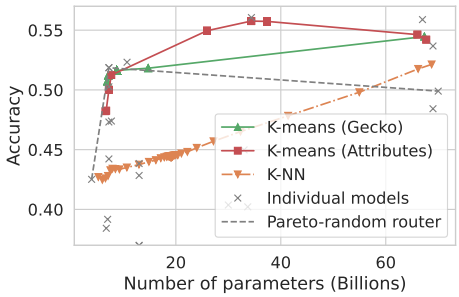
Over 400 independent trials, we randomly partition the data into 4000, 400, 5600 examples for training, validation, and testing, respectively. Unlike other datasets, we do *not* use Gecko for embedding prompts for Headlines. Rather, the frozen text embedding φ is constructed based on the output of the Gemma 2B model[1]: for each prompt, its embedding is the seven equally spaced quantiles of the per-token probabilities of the output tokens from Gemma 2B.

The mean deferral curves over 400 trials are shown in Figure 6c. We observe that our proposed `UniRoute` (LearnedMap) has higher accuracy than K-NN throughout the whole cost range.

Math+Code The Math+Code dataset is from Dekoninck et al. (2025) (containing a subset of Minerva Math and LiveCodeBench (Lewkowycz et al., 2022; Jain et al., 2024)). We use all 4 LLMs for testing; consequently, the training data is unlabeled, meaning that we cannot train `UniRoute` (LearnedMap) in this case. The parameter tuning procedure (Appendix G.1) for K cannot be applied to the Math+Code dataset, where we have no training LLMs. In this case, we set K to $\sqrt{N_{\text{val}}}$ for K-NN and to $N_{\text{val}}/50$ for K-Means.

[1]Gemma 2B model: https://huggingface.co/google/gemma-2b.

G.4 Train on Chatbot Arena and Test on EmbedLLM



Method	Area↑	QNC↓	Peak Acc.↑
Pareto-random router	.507	∞	51.9%
K-NN	.472	∞	52.1%
K-means (Gecko)	.529	∞	54.5%
K-means (Attributes)	.545	.97	55.8%

Figure 7: Deferral curves of routers trained on Chatbot Arena with pairwise comparison labels, and tested on EmbedLLM with per-prompt correctness labels.

We observe that representing each prompt with a small number of binary attributes that capture its inherent hardness shines when there is a prompt distribution shift at test time. To illustrate this, we use the seven binary difficulty attributes proposed in Li et al. (2024b), and prompt Gemini 1.5 Pro to annotate each attribute for each training prompt. We then construct a query embedder

$$\boldsymbol{\varphi}(\boldsymbol{x}) = \sigma(\boldsymbol{V}^\top \mathrm{Gecko}(\boldsymbol{x})) \in [0,1]^7, \tag{23}$$

where $\boldsymbol{V} \in \mathbb{R}^{768\times 7}$ is distilled using the training set to predict the 7-category attributes for any new prompt $\boldsymbol{x}$.

We compare Gecko-based prompt representation and attribute-based representation by training on Chatbot Arena conversation data (Zheng et al., 2023), and testing on EmbedLLM, which contains mostly Q&A prompts. To reduce confounding factors, we train on all LLMs that are present in both datasets (26 LLMs), and test on the same set of LLMs (i.e., no unseen LLMs at test time). After appropriate filtering, the Chatbot Arena dataset has 8447 records left. The filtering step ensures that we only deal with LLMs that are present in both datasets. These examples are split further into 90% training and 10% validation splits.

The Chatbot Arena dataset contains pairwise comparison labels: each user prompt is responded to by two random LLMs, to which the user selects the better response. To evaluate per-cluster performance for representing each LLM, we fit the Bradley-Terry-Luce model (Bradley & Terry, 1952) to the pairwise comparison labels within a cluster and estimate the pointwise quality scores for each LLM for that cluster. We use the full EmbedLLM dataset for testing.

The results are shown in Figure 7 where we compare two variants of our proposed `UniRoute` (K-means):

1. *K-means (Gecko).* This is `UniRoute` (K-means) (see Section 5.1) where the text embedding is set to the Gecko model (Lee et al., 2024b).
2. *K-means (Attributes).* This is `UniRoute` (K-means) where the text embedding is set to (23).

We observe that K-means (Attributes) in this case performs better than K-means (Gecko), suggesting that using prompt hardness attributes helps improve robustness to prompt distribution shifts. In fact, this routing approach is the only method that can reach the performance of the most accurate model in the pool, thus attaining a finite quality-neutral cost (QNC). The reason the Pareto-random router has a decreasing trend is because the Pareto-optimal LLMs are chosen using the validation set, and turn out to be not optimal for the test set.

G.5 Static LLM Pool Setting

While the dynamic pool setting is the focal point of our work, we show in Table 1 that even in the *static* LLM pool setting, `UniRoute` is typically comparable to most baselines. The MLP baseline alone often has a slight edge. This is because, unlike our methods, which are tied to a particular input LLM representation, the MLP approach has the flexibility of learning its own representation for each fixed LLM. Note that this is possible only in the static LLM pool setting.

In this case, all LLMs are seen during training, and hence the training and validation samples have correctness label annotations from all LLMs. We tune the hyper-parameters, such as K in K-NN and K-Means, the number of hidden nodes in MLP, and the intermediate dimension in Matrix Factorization, to maximize the area under the deferral curve on the validation sample. We pick K from the range $\{5, 10, 25, 50, 75, 100, 150, 200, 250, 300\}$ for K-NN and K-Means, pruning out values that are too large for the given validation sample size.

For `UniRoute` (LearnedMap), we used two hidden layers, with the same chosen number of hidden nodes as MLP. We set the number of clusters to be the same as the chosen number of clusters for `UniRoute` (K-Means). We employed Adam with learning rate 0.001 and batch size 64 for training, picking the checkpoint with the best quality metric on the validation set.

For these experiments, we additionally consider the Headlines dataset (see Appendix G.3) where all LLMs are fully observed during training. We additionally consider the LiveCodeBench coding benchmark (Jain et al., 2024), and include from it 15 LLMs for which the model size was publicly available (the Math+Code dataset contains two LLMs from this benchmark). We split the LiveCodeBench dataset into 60% for training, 10% for validation, and 30% for testing. For Headlines, we use 40%, 10%, 50% for training, validation, and testing, respectively.

Table 1: Comparing `UniRoute` with existing methods for the **static LLM pool** setting, where $\mathcal{H}_{\text{tr}} = \mathcal{H}_{\text{te}}$. We report the area under the deferral curve (↑). The best baseline results are highlighted, and the best `UniRoute` results are **boldfaced**. Even in the static setting, our approach is competitive compared to most baselines. The MLP baseline often has a slight edge. This is because, unlike our methods, which are tied to a particular input LLM representation, the MLP approach has the flexibility of learning its own representation for each fixed LLM. Note that this is possible only in the static LLM pool setting, and not the dynamic setting, which is the focus of this paper.

Method \ Dataset	EmbedLLM	MixInstruct	RouterBench	Math+Code	LiveCodeBench	Headlines
ZeroRouter (Hu et al., 2024b)	.601	.0483	.707	.392	.457	.834
MLP Hu et al. (2024b)	.689	.0500	.747	.483	.480	.852
Matrix Factorization (Ong et al., 2025; Zhuang et al., 2024)	.682	.0503	.744	.482	.482	.849
K-NN Hu et al. (2024b); Shnitzer et al. (2023)	.636	.0510	.744	.475	.474	.854
`UniRoute` (K-Means)	.682	**.0504**	.744	**.480**	**.481**	.845
`UniRoute` (LearnedMap)	**.683**	.0502	**.744**	.463	.479	**.854**

G.6 Visualisation of LLM Embeddings

We visualise the cluster-based LLM embeddings $\mathbf{\Psi}$ learned from our clustering procedure on the EmbedLLM and RouterBench datasets in Figures 8 and 9. These heatmaps illustrate the Gaussian kernel similarity between all pairs of LLM embeddings on these datasets. The results are largely intuitive: e.g., on RouterBench, we find that the `claude` family of models are highly similar to each other. Similarly, on EmbedLLM, code-focussed models generally tend to demonstrate a high degree of similarity.

We emphasise again that EmbedLLM previously also considered representing LLMs as feature vectors. Importantly, however, their representation depends on a *fixed* pool of LLMs, and does not readily generalise to new LLMs (without further retraining).

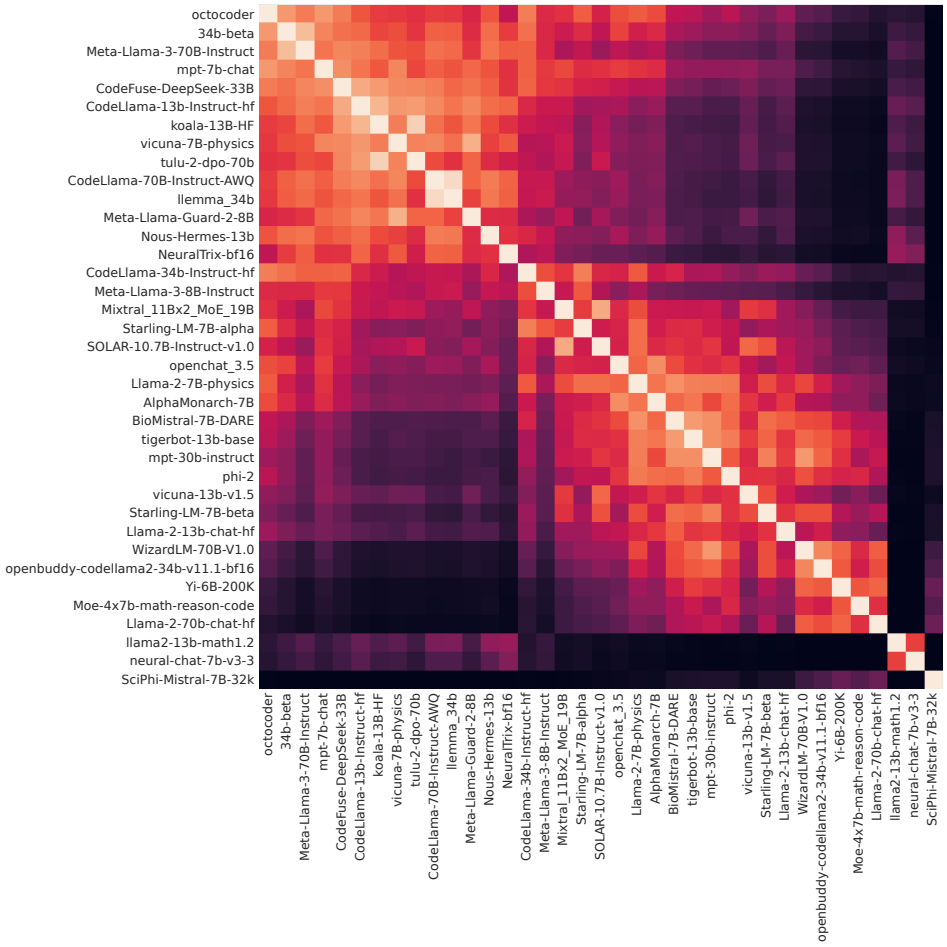


Figure 8: Gaussian kernel similarity between pairs of LLM embeddings on EmbedLLM dataset.

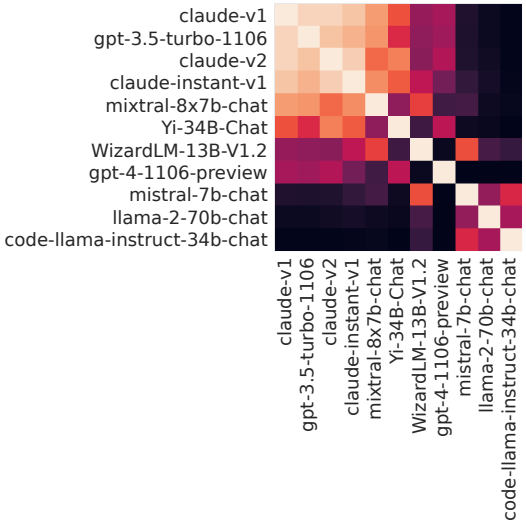


Figure 9: Gaussian kernel similarity between pairs of LLM embeddings on RouterBench dataset.

H COMPARISON TO ROUTER RETRAINING

H.1 CHALLENGES WITH ROUTER RETRAINING

Per the discussion in the body, naïve static router retraining is a conceptually appealing approach to the dynamic routing problem. While there may be certain specific scenarios where this approach is indeed advisable, we now argue why for many settings, it may fall short.

First, the central requirement in router retraining is the existence of a *labelled sample* of (prompt, target) pairs, along with the outputs of any new LLM. If this set of prompts is small – which is the setting considered in `UniRoute`, with the validation set S_{val} – then retraining runs the risk of overfitting. On the other hand, if this set of prompts is large, then the *annotation cost* can be cumulatively prohibitive: *every* new LLM will need to perform inference on the large prompt set. Further to this, the cost of retraining can be non-trivial, as it typically involves many steps of gradient-based optimisation (see the next section). thus potentially delaying availability of the new router.

Second, regardless of the size of the labelled prompt set, retraining and redeployment can involve a non-trivial engineering cost, as model redeployment is rarely *fully* automated. Third, retraining may fail to adapt to certain bespoke settings. For example, suppose that we wish to allow users to select a *subset* of models to route amongst (e.g., to enable consistency with their existing workflows). Supporting an arbitrary subset of *dynamic* models would naïvely require an exponential number of retraining rounds. By contrast, `UniRoute` would seamlessly handle such settings without any retraining.

H.2 THEORETICAL ANALYSIS OF ROUTER RETRAINING COSTS

We analyse the runtime complexity of `UniRoute` (K-Means) and a static routing method (such as "MLP (Clairvoyant)"; see Appendix G.2) that retrains from scratch on the full training set, whenever the LLM pool changes. We assume that the static routing method is a feedforward network given by a dense layer on top of a frozen query embedding network. This assumption agrees with our experimental setup.

For this analysis, we use the following notation.

Symbol	Description
N	Training sample size
N_{val}	Validation sample size used to represent LLMs in UniRoute. Note that $N_{\text{val}} \ll N$.
T	Number of training epochs
I	Maximum number of iterations in the K-means algorithm
M	Number of LLMs in the current pool
K	Number of clusters in UniRoute. Note that $K < N_{\text{val}}$.
D	Embedding dimension of each query
B	Cost for computing the embedding of one query.

Cost of router training

- `UniRoute` (K-Means): $O(BN + DNIK)$, which is the cost of running K-Means once on the training sample. In our experiments, we use $I = 300$ iterations. Note that the complexity is **independent of the LLM pool size** M.
- *Static router*: $O(BN + DNMT)$, where we assume that the cost of a gradient update on one example is $O(DM)$. Note that in the more general setup where the query embedding network is also trained jointly with the routing module, the gradient update cost will depend on the size of the embedding network, and the cost will be much higher.

Cost of adding a new LLM Suppose that the router has been trained. Consider the scenario where a new test LLM arrives at test time. Let C be the cost of calling the new LLM for generating a response for one prompt.

- `UniRoute` (K-Means): $O(CN_{\text{val}} + BN_{\text{val}} + DN_{\text{val}}K)$. The first term $O(CN_{\text{val}})$ is the cost of generating responses for all prompts in the validation set. The second term is for generating responses for prompts in the validation set. The third term is for computing cluster assignments. Note that this is independent of M.
- *Static router*: $O(CN + BN + DNMT)$. The first term is for generating responses for all prompts in the training set.

Observe that there is a large difference in the costs of the two methods for adding a new LLM. In particular, the cost of adding a new LLM with `UniRoute` (K-Means) is independent of the training sample size N and the size of the current LLM pool M. By contrast, retraining a static router would clearly depend on these quantities.

Cost of performing LLM inference At test time, the runtime complexity for routing one input query is as follows:

- `UniRoute` (K-Means): $O(B + DK + M)$. The second term is the cost of determining the nearest centroid. The third term is the cost of applying the routing rule in (8).
- *Static router*: $O(B + DM)$.

H.3 Empirical Analysis of Router Costs

We empirically analyse the actual wall-clock time of various key operations of `UniRoute` and a retrained router.

We consider the same experimental setting as in Figure 2 for the EmbedLLM dataset. We compare the wall time of our proposed `UniRoute` (K-Means) to the strongest baseline, which is Retrained MLP in this case. Note that all approaches studied in this work (including `UniRoute`) start from embedding queries with the Gecko embedding network. So, to minimize the variance in time measurement, we pre-compute Gecko embeddings and exclude this step from the following results. The following table reports time in seconds.

Operation	`UniRoute` (K-Means)	Retrained MLP
Training	1.41 s	8.18 s
Route one test example	0.37×10^{-3} s	3.19×10^{-3} s
Add a new LLM to the pool	57.90×10^{-3} s	8.18 s

For adding a new LLM to the pool, we exclude the time to run batch inference to generate responses for the prompts in the validation set because this is a common step for both approaches.

We observe that `UniRoute` is faster than Retrained MLP for both training and inference. In fact, it is also better in terms of quality (see Figure 2). Since Retrained MLP does not support dynamic routing, adding a new LLM to the pool requires retraining the router (hence the same wall time for training and for adding a new LLM). By contrast, `UniRoute` only requires estimating per-cluster errors of the new LLM. Unlike the MLP approach, this does not require any expensive gradient updates. The above wall time is consistent with the runtime complexity analysis in Appendix H.2.

We make the following additional remarks.

1. We emphasize that the above K-means clustering uses a much larger dataset compared to MLP. This is because K-Means clustering (training) requires only unlabeled input queries, whereas the MLP router requires labeled queries. Requiring no labels allows one to easily afford a large set of training queries. In the above experiment, the `UniRoute` uses 18k unlabeled examples, whereas Retrained MLP uses 3k labeled (validation) examples. Recall from Section 4.1 that we consider the setting where the validation set (i.e., containing labels from test LLMs) is limited.
2. The above latency analysis is in an illustrative setting where the query embedding network (i.e., Gecko in this case) is frozen. In reality, we expect router retraining to take much longer if the query embedder is also trained. In such a case, Retrained MLP would incur a

significant cost every time the LLM pool changes. Our `UniRoute` approach would require backbone training only once.

3. Inference latency and training time are only one aspect of the overall cost when deploying a router. In practice, there are engineering and maintenance costs that one needs to take into consideration.
4. There are many applications where full router retraining is infeasible either because smaller (open-sourced) models are released frequently, or because the number of possible subsets of LLMs one can use is exponentially large. Even if the underlying LLMs are fixed, the test-time LLM pool can change frequently. In particular, the exact pool of LLMs available to a user may vary, depending on e.g., the availability of GPUs for serving, LLM licensing requirements, suitability to the user's particular task, etc. One can imagine use cases where custom-made LLMs of different sizes have to be updated frequently (e.g., LLM assistants for stock market news), and possibly on proprietary data (e.g., LLMs for internal documents of a large company).

I ADDITIONAL RELATED WORK

Routing approach	Candidate LLMs	Training signals	Works without task labels	Adaptive computation	Unseen models at test time	Reference
Smoothie	Any	Query embeddings. No label required.	✓	✘	✘	Guha et al. (2024a)
Cascading with token-level features	2	Pointwise evaluation.	✓	✓	✘	Gupta et al. (2024)
Summon the titans	2	Annotations from a teacher model.	✓	✓	✘	Rawat et al. (2021)
RouteLLM	2	Pairwise comparison metrics.	✓	✓	✘	Ong et al. (2025)
K-NN router	Any	Pointwise evaluation, query embeddings.	✓	✓	△	Hu et al. (2024b); Shnitzer et al. (2023)
GraphRouter	Any	Task information	✘	✓	✓	Feng et al. (2024)
Our proposal	Any	Pointwise evaluation, query clusters	✓	✓	✓	This work

Table 2: A qualitative comparison of recently proposed model routing approaches. Adaptive computation refers to the ability to trade quality for a reduced inference cost. △: The K-NN approach considered in Hu et al. (2024b); Shnitzer et al. (2023) is for a fixed pool of LLMs. However, the approach can be straightforwardly extended to support unseen models at test time (i.e., dynamic pool).

Model routing. Model routing has emerged as a simple yet effective strategy to lower LLMs' inference cost (Hendy et al., 2023; Hari & Thomson, 2023). Recent works have studied various strategies to learn a router, including training an explicit "meta-model" based on a neuronal network (Ding et al., 2024; Šakota et al., 2024; Chen et al., 2024b; Aggarwal et al., 2024), k-nearest neighbours (Hu et al., 2024b; Shnitzer et al., 2023; Stripelis et al., 2024; Lee et al., 2024a), matrix factorisation (Ong et al., 2025; Zhuang et al., 2024; Li, 2025), and graph neural networks (Feng et al., 2024). Works have also explored the role of supervision in training a router (Lu et al., 2024; Zhao et al., 2024b), and enhancing router robustness (Dann et al., 2024; Montreuil et al., 2025; Shafran et al., 2025). Typically, the router orchestrates amongst multiple independent LLMs, although it is also possible to route amongst *implicit sub*-models, such as those defined by a MatFormer (Devvrit et al., 2024; Cai et al., 2024a).

Model cascading. Cascading is a closely related technique for orchestrating amongst multiple models, wherein one *sequentially invokes* each model in order of cost. One then uses statistics from the resulting model output (e.g., the confidence) to decide whether or not to proceed to the next costlier model. Cascading has a long history in computer vision applications (Viola & Jones, 2001; Wang et al., 2018; Swayamdipta et al., 2018; Rawat et al., 2021; Wang et al., 2022; Kag et al., 2023; Jitkrittum et al., 2023). Recently, cascades have also been successfully proven in the case of LLMs (Varshney & Baral, 2022; Chen et al., 2023b; Gupta et al., 2024; Yue et al., 2024; Chen et al., 2024a; Nie et al., 2024; Chuang et al., 2025).

Selective classification and learning to defer. The formal underpinnings of routing and cascading can be traced to three closely related literatures: learning to reject (Chow, 1970; Bartlett & Wegkamp, 2008; Cortes et al., 2016), selective classification (Geifman & El-Yaniv, 2019; Narasimhan et al., 2024a;b), and learning to defer to an expert (Madras et al., 2018; Sangalli et al., 2023). Following pioneering studies of Trapeznikov & Saligrama (2013); Bolukbasi et al. (2017); Mozannar & Sontag (2020), a series of works have studied the routing and cascading problem through these lenses (Narasimhan et al., 2022; Mao et al., 2024a;b;c;d).

Model fusion Model routing may be contrast to model *fusion*, where the primary goal is to leverage multiple models to improve *quality*, potentially at the expense of *efficiency*. This can involve invoking multiple models prior to generating an output (Ravaut et al., 2022; Jiang et al., 2023; Guha et al., 2024b; Wang et al., 2024b; Hu et al., 2024a), or producing a single fused model (Singh & Jaggi, 2020).

Mixture of experts (MoE). Classically, MoE models focused on learning parameters for independent models, along with a suitable routing rule (Jacobs et al., 1991; Jordan & Jacobs, 1993). These have proven an plausible alternative to model specialisation (Jang et al., 2023; Douillard et al., 2024). Such models are typically of the same size; thus, cost considerations do not factor into the router design. More recently, MoEs have focussed on *sub*-models within a single larger model, e.g., a Transformer (Fedus et al., 2022; Zhou et al., 2022).

Early-exiting. Early-exiting enables adaptive computation within a *single* neural model, by allowing computation to terminate at some intermediate layer (Teerapittayanon et al., 2016; Scardapane et al., 2020; Zhou et al., 2020). Often, the termination decision is based on a simple model confidence (akin to simple model cascading), but learning approaches have also been considered (Xin et al., 2020; Schuster et al., 2022).

Speculative decoding. Speculative decoding is another technique that leverages two models to speed up inference, by using the smaller model to draft tokens and having the larger model verify them (Stern et al., 2018; Chen et al., 2023a; Leviathan et al., 2023; Tran-Thien, 2023; Sun et al., 2024; Zhou et al., 2024a; Cai et al., 2024b; Li et al., 2024d;e). Recent works have studied approaches to combine speculative decoding with early-exits (Elhoushi et al., 2024) and cascades (Narasimhan et al., 2025).

Model representation. Broadly, the idea of representing models via compact representations has been studied in various contexts (Achille et al., 2019; Yan et al., 2020; Cotler et al., 2023). In the context of LLMs, our proposal relates to several recent strands of work (Tailor et al., 2024; Thrush et al., 2024; Zhuang et al., 2024; Feng et al., 2024; Li, 2025; Zhao et al., 2024b), which merit individual discussion.

The idea of representing an LLM via a prediction error vector has close relation to some recent works. In Tailor et al. (2024), the authors proposed to represent "experts" via predictions on a small *context set*, so as to enable deferral to a new, randomly selected expert at evaluation time. While not developed in the context of LLMs (and for a slightly different problem), this bears similarity to our proposal of representing LLMs via a prediction error vector on a validation set; however, note that the mechanics of routing based on this vector (via suitable clustering) are novel. Thrush et al. (2024) considered representing LLMs via their perplexity on a set of public benchmarks, for subsequent usage in pre-training data selection. This shares the core idea of using LLM performance on a set of examples to enable subsequent modelling, albeit for a wholly different task. Zhuang et al. (2024) proposed to construct a generic embedding for LLMs based on performance on public benchmarks. This embedding is constructed via a form of matrix factorisation, akin to Ong et al. (2025). While Zhuang et al. (2024) discuss model routing as a possible use-case for such embeddings, there is no explicit evaluation of embedding generation in the case of *dynamic* LLMs; note that this setting would naïvely require full re-computation of the embeddings, or some version of incremental matrix factorisation (Brand, 2002).

Some recent works have considered the problem of routing with a dynamic pool of LLMs. Feng et al. (2024) proposed a graph neural network approach, wherein LLMs are related to prompts and *tasks* (e.g., individual benchmarks that a prompt is drawn from). Such pre-defined *task labels* for input prompts may be unavailable in some practical settings.

Li (2025) proposed to use LLM performance on benchmark data to construct a *model identity vector*, which is trained using a form of variational inference. This has conceptual similarity with our prediction error vector proposal (and the works noted above); however, our mechanics of learning based on this vector (i.e., the cluster-based representation) are markedly different. We reserve a separate discussion of this work in Appendix I.1.

Finally, Zhao et al. (2024b) consider the problem of routing to a dynamic *LoRA pool*, where LoRA modules are natively represented by aggregating (learned) embeddings on a small subset of training examples. These embeddings are learned via a contrastive loss, constructed based on certain pre-defined task labels. As such labels may not be provided in many settings, Chen et al. (2024a) implemented a variant wherein these are replaced by unsupervised cluster assignments. While similar in spirit to our proposals, the details of the mechanics are different; e.g., we use the clustering to compute a set of average accuracy scores for each LLM, rather than training an additional embedding.

Finally, we note that Chen et al. (2024a) also consider the use of clustering as part of router training. However, the usage is fundamentally different: Chen et al. (2024a) regularise the learned prompt embedding $\mathbf{\Phi}$ based on cluster information, while we use clustering to construct the LLM embedding $\mathbf{\Psi}$.

Our proposed approach is similar in spirit to methods such as K-NN, where the prediction for a prompt is based on the labels associated with queries in its neighborhood; in our case, we consider pre-defined clusters instead of example-specific neighborhoods.

I.1 Comparison to Li (2025)

In this section, we contrast our proposed `UniRoute` to the the LLM Bandit approach of Li (2025), which is one of the few routing techniques relevant to the dynamic routing setting. *The two techniques are fundamentally different*.

Li (2025) consider an online routing setting wherein bandit approaches are advisable, whereas we consider and analyse a conventional supervised learning setting. Also of note is that the mechanics of producing an LLM embedding are different in their setting. Briefly, a model identity vector in their work is a parameter to a feedforward network that predicts the correctness of an LLM from a prompt. For each LLM, such a vector is optimized so that the network has low cross-entropy loss, regularized by a KL term to encourage the vector to be close to a standard Gaussian prior. Because this is an expensive procedure to add a new LLM to the pool, they propose an efficient version wherein each prompt is rated for its difficulty. The difficulty of a prompt is defined as the mean of cross entropy losses on the prompt across all LLMs in the current pool. They then apply a stratified sampling on the full prompt set, using this difficulty score as the strata. They run the same above procedure on the stratified sample to find model identity vectors.

In more detail: in Li (2025), the representation of each LLM in the pool is the parameter vector of a feedforward network. For each LLM h, the parameter is optimized so that the network can predict the correctness score $\mathbf{1}[\boldsymbol{y} \neq h(\boldsymbol{x})]$ of the LLM given an input prompt $\boldsymbol{x}$ (sec Section 5.2 for a related discussion). *Adding a new LLM thus involves an expensive gradient-based training procedure.* While Li (2025) also proposed an efficient procedure that requires only 50 prompts for training, the procedure involves counting the number of LLMs in the current pool that can correctly respond to each prompt in a training set. This means that the representation of the $(K+1)$-th LLM will necessarily depend on the K models in the current pool. In other words, *the order in which LLMs arrive in the pool influence the learned identity vectors of future LLMs*.

Our proposed LLM representation in (12) is computationally efficient: It does not require expensive gradient updates, does not depend on the LLM pool (see the runtime complexity analysis in Appendix H). In `UniRoute`, LLM representations are computed independently of each other (i.e., not a function of the order of LLMs).

Empirical comparison Since there is no source code released by Li (2025), we present a comparison against the numbers reported in Figure 3b of Li (2025) for the OpenLLMv2 dataset (Fourrier et al., 2024).[2] In this experiment, we consider the setting of using four Qwen 2.0 model variants as training LLMs, and six Qwen 2.5 models as test LLMs. See Table E.4 and Table E.5 in Li (2025) for details. The OpenLLMv2 dataset has 20046 examples in total, from which we use only 800 randomly selected examples as the validation set for `UniRoute`, 6014 examples for testing (30%), and the rest of 13232 examples for training. We report Quality-Neutral Cost (QNC) as in Figure 2 of our work.

[2] OpenLLMv2 dataset: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard

Method	QNC ↓
ZeroRouter	86.4%
K-NN	81.0%
Retrained MLP	84.1%
Retrained MatFac	81.0%
`UniRoute` (K-Means)	82.2%
`UniRoute` (LearnedMap)	80.2%
LLM Bandit (Li, 2025)	> 90%

In the above results, the QNC of LLM Bandit is directly estimated from Figure 3b in Li (2025) by considering the operating point of "PCDR (10)" with the highest evaluation score. More precisely, that operating point has a cost of approximately \$1.8. With the most performant model in the pool costing \$2.0, the QNC is at least \$1.8/\$2.0 = 0.9 = 90%. The results indicate that `UniRoute` is able to achieve the same performance as the largest model in the pool (Qwen2.5-72B-Instruct) with 80.2% of its cost.