INFERENCE-TIME MODEL SELECTION VIA CAPABILITY SYNTHESIS

Anonymous authors
Paper under double-blind review

ABSTRACT

We consider the problem of inference-time model selection in language models, where a selector has access to a finite pool of models and aims to select the model that performs best on a given task. In this setup, querying every model on each task is computationally expensive; therefore, the goal is to design contextual model selection strategies that can predict the best-performing model for each context. We propose a simple framework named *Capability Synthesis* that discovers directions along which the models' capabilities are most distinguishable. Capability Synthesis learns to jointly represent models and tasks in a shared capability space, and uses these representations to predict the relative performance of each model on any given task at inference time. This framework is particularly suited for model selection over large pools of models, as larger pools provide more observations for learning the capability structure, while the inference cost of the selection strategy remains negligible. Finally, our experiments on multiple routing benchmarks show that this approach, which admits closed-form solutions, outperforms more complex trainable routers and single-model deployments.

1 INTRODUCTION

Evaluations and benchmarks record the performance of a large collection of language models on tasks of various difficulties. These performance records provide information about the joint space of models and tasks, and where they stand relative to each other (Fourrier et al., 2024; Huang et al., 2025). This opens up the interesting opportunity to study the structure behind this joint space, and develop methods to better understand the landscape of the capabilities of language models and their downstream applications (Maimon et al., 2026; Ruan et al., 2024).

In this paper, we study the problem of inference-time model selection in language models. The main question we want to answer is whether we can learn a shared structure of capabilities where both models and tasks can be represented, and use this structure to guide model selection. In model selection, we have access to a finite pool of models, and the goal is to design an algorithm that can select the best model for a given task, without querying all the models on the task and observing their performance. This is also known as the routing problem, and several papers have studied algorithms and architectures for routers in language models (Shnitzer et al., 2023; Chen et al., 2023; Zhuang et al., 2025; Zhang et al., 2025b; Moslem & Kelleher, 2026).

We propose a simple framework, named Capability Synthesis, that uses the performance matrix of the models on a set of tasks to learn a shared space of capabilities and derive closed-form representations for models and tasks in this space. Capability synthesis consists of a principal component analysis (PCA) step to learn the capability profiles and task specifications, and a regression step to learn a synthesizer that generalizes the task specifications to any task that the router may receive at inference. We show a simple procedure to extend the pool of models with minimal refitting cost.

We evaluate Capability Synthesis on large-scale routing benchmarks (Huang et al., 2025; Zhuang et al., 2025) and compare it to several routing and model selection baselines. Our empirical results show that simple model selection algorithms can compete with and outperform more complex routers, particularly when the pools have a large number of models. Finally, we categorize three oracle comparators based on the information they can observe before selection, and explain how much of the gap between single model deployment, and ensemble strategies can be explained by model selection.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We summarize our contributions as follows:

- We propose Capability Synthesis that derives a closed-form representation of models and tasks in a shared capability space, and design a contextual model selection algorithm that uses these representations for inference-time model selection in language models.
- The training of the router doesn't require any gradient updates; new models can be added to the pool with minimal refitting, and inference cost is low. This makes this strategy suitable for model selection over large pools of models.
- We evaluate Capability Synthesis on various routing benchmarks, and compare it with state-of-the-art routing and model selection strategies. Our results show that as the size of the pool grows, capability synthesis outperforms the baselines and maintains a consistent gain over the best single model.
- We analyze the gap between the best single-model deployment and an oracle that can verify the generation of every model before selection. We show that part of this gap can be explained by contextual model selection, and the remaining part is the gain of verification at inference.

2 Related Work

Prior literature has studied various routing techniques and shown the gain of model selection over single model deployments. for Recent clustering-based routers Zhang et al. (2025b) have shown that a well-curated pool paired with a good routing strategy can compete with and outperform frontier models.

Several routing methods train an encoder for learning representations of models and their capabilities. EmbedLLM (Zhuang et al., 2025) uses an encoder-decoder architecture to learn general-purpose embeddings for models and tasks, which can be utilized for benchmark performance prediction and routing. Model-SAT router (Zhang et al., 2025a) is a model selection framework that uses capability instruction tuning for model selection, that train an encoder to represent model capabilities, and design an aptitude test for adding new models to the pool.

Other routers use contrastive methods for routing and model selection. RouterDC (Chen et al., 2024) proposes a contrastive learning method to address a common failure of routers when multiple LLMS succeed on queries. Cost-aware contrastive routing (Shirkavand et al., 2026) maps models and prompts into a shared embedding space and train a contrastive encoder for routing.

Several methods study routing for cascades such as HybridLLM (Ding et al., 2024) and FrugalGPT (Chen et al., 2023). RouteLLM (Ong et al., 2025) uses human preference data to dynamically select between a strong and weak LLM during inference to balance performance and cost.

Finally, model selection with benchmark datasets has been studied by early on (Shnitzer et al., 2023). Apart from routing literature, estimating capability directions using PCA on benchmark matrices has been studied by (Ruan et al., 2024), where they used these results to derive scaling laws for language models and predict emergent capabilities with scale.

3 Capability Synthesis

3.1 Problem Statement

Denote $\mathcal{X}$ as the space of input queries, and $\mathcal{Y}$ as the space of responses. Consider a finite pool of M candidate models $\Pi = \{\pi^i\}_{i=1}^M$, where $\forall i \in [M], \pi^i : \mathcal{X} \to \mathcal{Y}$. The models $\pi \in \Pi$ are language models that might differ in scale, expertise, or alignment. To evaluate the quality of a generated response $y \sim \pi(\cdot \mid x)$, we have access to a verifier $\mathcal{V} : \mathcal{X} \times \mathcal{Y} \to [0, 1]$ that scores a given query-response pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$.

At inference time, the model selection algorithm has access to the pool $\Pi = \{\pi^i\}_{i=1}^M$, and given task $x \in \mathcal{X}$, the goal is to select the model that has the best performance on x,

$$i_x^\star = \arg\max_{i \in [M]} \mathbb{E}_{y^i \sim \pi^i(\cdot|x)} \left[\mathcal{V}(x, y^i) \mid x \right]$$

Notation: For any $n \in \mathbb{N}$, denote $[n] = \{1, \ldots, n\}$. We will consistently use the superscript π^i to denote the model index, and the subscript x_j to denote other indices. For a task $x \in \mathcal{X}$, we denote the task embedding $\mathbf{x} = \phi(x) \in \mathbb{R}^d$, where $\phi : \mathcal{X} \to \mathbb{R}^D$ is a pretrained embedding model. For clarity of notation, we will use bold $\mathbf{x}$ to denote the d-dimensional embedding and drop ϕ unless it's not obvious from the context. We will use x or $\mathbf{x}$ interchangeably to refer to the task, but x consistently denotes the task description and $\mathbf{x}$ is the task embedding vector.

3.2 METHOD

Consider the pool of models $\Pi = \{\pi^i\}_{i=1}^M$, and a set of calibration tasks $\{x_j\}_{j=1}^N \subseteq \mathcal{X}$. Capability Synthesis operates upon a performance matrix $R \in \mathbb{R}^{M \times N}$, where the (i, j)'th element is

$$R(i, j) = r_i^j, \qquad r_i^j = \mathcal{V}(x_j, \pi^i(\cdot \mid x_j))$$

The goal in capability synthesis is to discover directions towards which we get maximum pairwise model separation, and jointly represent models and tasks in the shared capability space spanned by these directions.

Formally, we want to find orthonormal vectors $\{\mathbf{v}_k\}_{k=1}^d \subseteq \mathbb{R}^N$, where $1 \leq d \leq \min(M, N)$, and $\mathbf{v}_k \in \mathbb{R}^N$ is the k'th direction that maximizes the average pairwise model separation,

$$\mathbf{v}_k = \underset{\substack{\|\mathbf{v}\|=1 \\ \mathbf{v} \perp \langle \mathbf{v}_1, \ldots, \mathbf{v}_{k-1} \rangle}}{\arg\max} \frac{1}{2M^2} \sum_{i=1}^{M} \sum_{l=1}^{M} \left[(\mathbf{r}^i - \mathbf{r}^l)^T \mathbf{v} \right]^2$$

where $\mathbf{r}^i \in \mathbb{R}^N$ is the i'th row of the performance matrix R. The unit vectors $\{\mathbf{v}_k\}_{k=1}^d$ are d capability directions along which the models' capabilities are most distinguishable. We will refer to the d-dimensional space spanned by these directions $\langle \mathbf{v}_1, \ldots, \mathbf{v}_d \rangle \subseteq \mathbb{R}^N$ as the capability space.

For model selection, we will formally derive the following representations using the performance matrix and task representations, and show how the relative performance can be recovered from their dot product:

1. **Capability Profile** $\mathbf{p} : \Pi \to \mathbb{R}^d$: Given model $\pi^i \in \Pi$, the capability profile $\mathbf{p}^i \in \mathbb{R}^d$ represents model π^i in the capability space. The element $\mathbf{p}_k^i \propto (\mathbf{p}^i)^T \mathbf{v}_k$ quantifies the capability of model π^i towards direction k.
2. **Synthesizer** $\mathbf{q} : \mathcal{X} \to \mathbb{R}^d$: For any query $x \in \mathcal{X}$, the mapping $\mathbf{q}(x) \in \mathbb{R}^d$ is the task specification in each capability direction, which quantifies what combination of capabilities is required to solve task x.

The capability Synthesis method consists of two phases. In the first phase, we perform Principal Component Analysis (PCA) on the performance matrix $R \in \mathbb{R}^{M \times N}$. Similar to the standard PCA procedure, we first subtract the column mean from each element of the column,

$$\tilde{R} = \left(\mathbf{I}_m - \frac{1}{M} \mathbf{1}^T \mathbf{1} \right) R$$

and then perform Singular Value Decomposition (SVD),

$$\tilde{R} \approx U_d \Sigma_d V_d^T = \left(U_d \sqrt{\Sigma_d} \right) \left(V_d \sqrt{\Sigma_d} \right)^T = P Q^T$$

where,

$$V_d = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_d \end{bmatrix} \subseteq \mathbb{R}^{N \times d}$$

$$P = U_d\sqrt{\Sigma_d} = \begin{bmatrix} \mathbf{p}^1 \\ \mathbf{p}^2 \\ \vdots \\ \mathbf{p}^M \end{bmatrix} \subseteq \mathbb{R}^{M\times d}, \qquad Q = V_d\sqrt{\Sigma_d} = \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_N \end{bmatrix} \subseteq \mathbb{R}^{N\times d}$$

Standard PCA chooses the singular directions $\{\mathbf{v}_k\}_{k=1}^d$ by finding the top d eigenvectors of the covariance matrix $\Sigma = \frac{1}{M}\tilde{R}^T\tilde{R}$. Formally, for $k \in [d]$,

$$\begin{aligned}
\mathbf{v}_k &= \underset{\substack{\|\mathbf{v}\|=1 \\ \mathbf{v}\perp\langle \mathbf{v}_1,\ldots,\mathbf{v}_{k-1}\rangle}}{\arg\max} \frac{1}{M} v^T\tilde{R}\tilde{R}v = \underset{\substack{\|\mathbf{v}\|=1 \\ \mathbf{v}\perp\langle \mathbf{v}_1,\ldots,\mathbf{v}_{k-1}\rangle}}{\arg\max} \frac{1}{M}\sum_{i=1}^{M}\left[(\tilde{\mathbf{r}}^i)^T\mathbf{v}\right]^2 \\
&= \underset{\substack{\|\mathbf{v}\|=1 \\ \mathbf{v}\perp\langle \mathbf{v}_1,\ldots,\mathbf{v}_{k-1}\rangle}}{\arg\max} \frac{1}{2M^2}\sum_{i=1}^{M}\sum_{l=1}^{M}\left[(\tilde{\mathbf{r}}^i - \tilde{\mathbf{r}}^l)^T\mathbf{v}\right]^2 \\
&= \underset{\substack{\|\mathbf{v}\|=1 \\ \mathbf{v}\perp\langle \mathbf{v}_1,\ldots,\mathbf{v}_{k-1}\rangle}}{\arg\max} \frac{1}{2M^2}\sum_{i=1}^{M}\sum_{l=1}^{M}\left[(\mathbf{r}^i - \mathbf{r}^l)^T\mathbf{v}\right]^2
\end{aligned}$$

where $\tilde{\mathbf{r}}^i = \mathbf{r}^i - \bar{\mathbf{r}}$ and $\bar{\mathbf{r}} = \frac{1}{M}\sum_{i=1}^{M}\mathbf{r}^i$. We will use $\mathbf{p}^\mathbf{i} \in \mathbb{R}^d, i \in [M]$ as the capability profile of model $\pi^i \in \Pi$, and $\mathbf{q}_j \in \mathbb{R}^d, j \in [N]$ as the synthetic core capabilities of task x_j. Finally, in order to learn the functional $\mathbf{q} : \mathcal{X} \to \mathbb{R}^d$, we will use the labeled dataset $D_{cal} = \{(\mathbf{x}_j, \mathbf{q}_j)\}_{j=1}^N$, and learn synthesizer parameters $\theta \in \mathbb{R}^{D\times d}$ via Ridge Regression,

$$\hat{\theta} = \arg\min_\theta \|X\theta - Q\|_F^2 + \lambda\|\theta\|_F^2 \tag{1}$$

where $X \subseteq \mathbb{R}^{N\times D}$ is the query embeddings $\{\mathbf{x}_i\}_{i=1}^N$ stacked as matrix. Note that, given λ, the Ridge Regression objective 1 has a closed-form solution.

Given capability profiles and the synthesizer from Capability Synthesis, the inference-time model selection algorithm receives task $x \in \mathcal{X}$ at inference and computes task specification $\mathbf{q}(x) = \hat{\theta}^T$ using the synthesizer. The method then selects the model $\hat{i}_x = \arg\max_{i\in[M]} \mathbf{q}(x)^T\mathbf{p}^i$, and queries the selected model on task x. Note that, $\mathbf{q}(x)^T\mathbf{p}^i$ recovers the relative performance of model $i \in [M]$ on task x with respect to the model pool, and the model with the highest estimated relative performance is the final selected model.

3.3 Model Pool Extension

Consider a setup where Capability Synthesis has been fitted on a performance matrix $R \in \mathbb{R}^{M\times N}$ of the pool $\Pi = \{\pi^i\}_{i=1}^M$, and we want to add a new model π^{M+1} to the pool. We show how adding a new model to a learned capability space doesn't require refitting the synthesizer $\mathbf{q}(x)$ and existing capability profile $\{\mathbf{p}^i\}_{i=1}^M$, unless the new model π^{M+1} has a capability that previously didn't exist in the pool.

We start by centering the reward vector of the new model π^{M+1} using the previous mean, $\tilde{\mathbf{r}}^{M+1} = \mathbf{r}^{M+1} - \bar{\mathbf{r}}$, and then estimate the capability profile of π^{M+1} in the learned capability space,

$$\mathbf{p}^{M+1} = \arg\min_{p\in\mathbb{R}^d} \|\tilde{\mathbf{r}}^{M+1} - Qp\| = \Sigma_d^{-1/2}V_d^T\tilde{\mathbf{r}}^{M+1}$$

Finally, we compute the residual and run a criterion to check whether the current capability space captures the capability directions added by the new model up to some tolerance threshold.

Note that, centering $\mathbf{r}^{M+1}$ with the previous mean $\bar{\mathbf{r}}$ doesn't affect selection, as the selection strategy 2 is invariant to the offset vector $\mathbf{b} \in \mathbb{R}^N$ that is common to all models. If every profile is computed as $\Sigma_d^{-1/2}V_d^T(\mathbf{r}^i - \mathbf{b})$, then $\hat{i}_x$ does not depend on $\mathbf{b}$.

We summarized the pool extension procedure with a heuristic refit criteria in Algorithm 3 in Appendix A.1.

Algorithm 1: Capability Synthesis (via PCA)

Input: Model Pool $\Pi = \{\pi^i\}_{i=1}^M$, Calibration Dataset $\{\mathbf{x}_j\}_{j=1}^N$, Performance Matrix $R \in \mathbb{R}^{M \times N}$, Regularization parameter $\lambda \in \mathbb{R}$, Rank $d \in [\min(M, N)]$

Output: Capability profiles $\{\mathbf{p}^i\}_{i=1}^M \subseteq \mathbb{R}^d$, Synthesizer $\mathbf{q} : \mathcal{X} \to \mathbb{R}^d$

// **Phase 1. PCA**

1

$$\tilde{R} = \left(\mathbf{I}_m - \frac{1}{M}\mathbf{1}^T\mathbf{1}\right) R$$

$$\tilde{R} \approx U_d \Sigma_d V_d^T = \left(U_d\sqrt{\Sigma_d}\right)\left(V_d\sqrt{\Sigma_d}\right)^T = PQ^T$$

// **Phase 2. Ridge Regression**

2

$$\hat{\theta} = \arg\min_{\theta} \|X\theta - Q\|_F^2 + \lambda\|\theta\|_F^2, \qquad \text{where} \quad X = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_N \end{bmatrix}$$

3 **return** $\{\mathbf{p}^i\}_{i=1}^M$, $\mathbf{q}(x) := \hat{\theta}^T\mathbf{x}$

Algorithm 2: Inference-Time Model Selection

Input: Model Pool $\Pi = \{\pi^i\}_{i=1}^M$, Capability profiles $\{\mathbf{p}^i\}_{i=1}^M$, Synthesizer $\mathbf{q} : \mathcal{X} \to \mathbb{R}^d$

Output: Final response y_x, Selected Index $\hat{i}_x$

1 Receive query x,

2 Select model $\hat{i}_x = \arg\max_{i \in [M]} \mathbf{q}(x)^T\mathbf{p}^i$

3 In case of a tie, select the model with lower estimated cost

4 Generate response $y \sim \pi^{\hat{i}_x}(\cdot|x)$

5 **return** y_x, $\hat{i}_x$

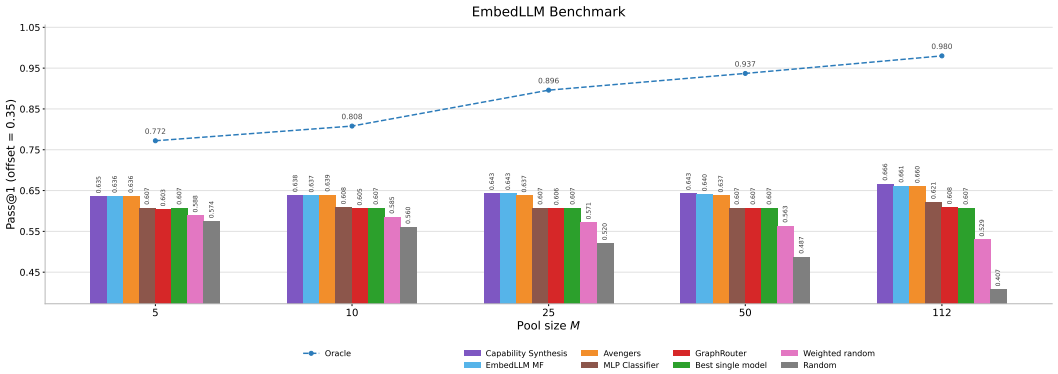
4 EXPERIMENTS

Routing Benchmarks We evaluate the performance of capability synthesis on the following routing benchmarks,

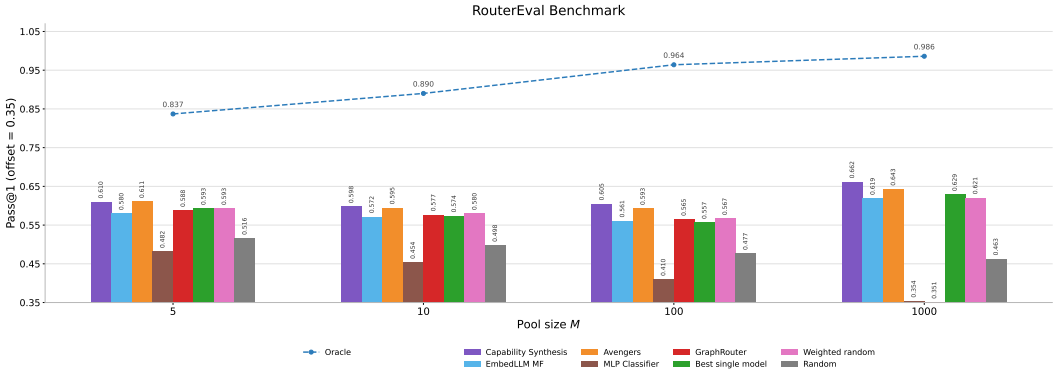
- **RouterEval** (Huang et al., 2025) contains over 200 million model-task performance record of the evaluation of LLMs on 12 benchmarks. For each benchmark, they construct two types candidate pools (easy pools, and hard pools) to test the performance of routers on different types of pools.
- **EmbedLLM** (Zhuang et al., 2025) contains the performance trace of 112 open-source LLMs on 36054 tasks in the form of a dense binary reward matrix, which we use as our performance matrix R.

We chose these benchmarks because they are suitable for evaluating the performance of model selection algorithms on pools of increasing size.

Baselines We compare capability synthesis with eight routing baselines: Avengers (Zhang et al., 2025b) which is a clustering-based method; GraphRouter (Feng et al., 2025) that builds a graph of models and queries, and predicts the performance of query-model edges; and EmbedLLM MF, which trains a router with an encoder-decoder architecture (Zhuang et al., 2025). We train a simple MLP Classifier with one hidden layer and ReLU activation that learns to predicts the index of the selected model from task embeddings. We include non-contextual routers such as (uniform) random selection and weighted random selection suggested by Zhuang et al. (2025). The weighted



(a) EmbedLLM Benchmark with pools of size $M \in \{5, 10, 25, 50, 112\}$.



(b) RouterEval Benchmark with pools of size $M \in \{5, 10, 100, 1000\}$.

Figure 1: Comparison of Capability Synthesis versus Baselines. The y-axis is the Pass@1 score on test data, and the x-axis is the size of the model pool M. The dashed line is the performance of the realized reward oracle.

random selection samples models with probabilities proportional to the capability Synthesis method selections. Finally, we include the best single model and the oracle as two important comparators.

For the embedding model, we use all-mpnet-base-v2 (Song et al., 2020) for EmbedLLM, and RoBERTa (768-dim) (Liu et al., 2019) for RouterEval. For EmbedLLM, we use the weak-to-strong pools, and for RouterEval, we report the mean over 12 datasets and 3 candidate pool types. The d, and λ parameters for capability synthesis, as well as the number of clusters k for Avengers, are chosen by hyperparameter sweep on validation split of the data. Detailed configurations and hyperparameters of all the methods is available in Appendix A.2.3.

4.1 Results

The results are summarized in Figure 1. We report pass@1 versus the size of the model pool on RouterEval and EmbedLLM benchmarks. [1] Since the rewards are binary in these benchmarks, pass@1 score is equal to the test accuracy of routers. We observe that Capability Synthesis, Avengers, and EmbedLLM MF have very close performance and outperform other routers in both benchmarks. As the size of the pool M grows, Capability Synthesis is able to take the lead and outperform other baselines.

Increasing the size of the pool adds more complexity to routing; however, scaling M can help to improve routing in capability synthesis, because more models provide more observations to better estimate the capability profiles $\mathbf{p}^i, i \in [M]$ and task specifications $\mathbf{q}_j, j \in [N]$.

[1] Note that the benchmarks only report a single generation and reward for each model-query pair; therefore, error bars of performance can not be computed for the baselines.

Oracles	Observed Info before Selection	Expected Performance
Expected Reward Oracle (Best Single Deployment)	$\varnothing$	$J^{ER} := \max_{i\in[M]} \mathbb{E}_x\Big[\mathbb{E}_{y^i\sim\pi^i(\cdot\|x)}\big[\mathcal{V}(x,y^i)\,\big|\,x\big]\Big]$
Model Selection Oracle (Contextual Routing)	$\sigma(x)$	$J^{MS} := \mathbb{E}_x\Big[\max_{i\in[M]} \mathbb{E}_{y^i\sim\pi^i(\cdot\|x)}\big[\mathcal{V}(x,y^i)\mid x\big]\Big]$
Realized Reward Oracle (Ensemble Methods)	$\sigma\left(x, r_x^1, \ldots, r_x^M\right)$	$J^{RR} := \mathbb{E}_x\Big[\mathbb{E}_{y^i\sim\pi^i(\cdot\|x)}\big[\max_{i\in[M]} \mathcal{V}(x,y^i)\mid x\big]\Big]$

Table 1: Categorizing Oracles based on the information they can observe before Model Selection at inference time, and their expected downstream performance. Here, $\sigma(Z)$ denotes the sigma-algebra of variable Z.

We've also included the performance of the best single model and the oracle in these plots. Among the routers, Capability Synthesis, Avengers, and EmbedLLM are able to consistently outperform the best single model. GraphRouter has almost the same performance of the best single model, except that on the largest pool ($M = 1000$) of RouterEval, it collapses to 0.35, which is below random selection.

Our empirical results also confirm the remaining gap to oracle discussed by recent literature on routing (Li et al., 2026; Chen, 2026). In section 5, we provide theoretical insights into why part of this gap is merely a gain of verification and can not be explain by model selection. We characterize three oracles based on the information they can observe before selection at inference time, and argue that the oracle observes both the context and realized reward, whereas the contextual routers, including capability synthesis, only observe the context. Therefore, the oracle that we are competing against has access to the gain of verification that contextual routers do not have.

Detailed numerical results for pass@1 and regret of each baseline and oracles are available in Appendix A.2.2.

5 The Gap to Oracle

For a given pool $\Pi = \{\pi^i\}_{i=1}^M$, one can consider different deployment strategies at inference,

1. The first strategy is fixed-model deployment where we evaluate all the models $\pi \in \Pi$ on an offline dataset and only deploy the model that achieves the highest average performance on the dataset at inference time This approach spends no inference time compute for model selection, as the selected model is fixed for all $x \in \mathcal{X}$. We will refer to this comparator that always select the best average model, and do not observe (or disregards) the task x before model selection, as the *expected reward oracle*.

2. The second deployment strategy is to treat the pool as an ensemble, query all the models on the task at inference time $y^i \sim \pi^i(\cdot \mid x)$, observe the reward from the verifier $r_x^i \sim \mathcal{V}(x, y^i)$, and pick the model that has the highest realized reward $i_x = \arg\max_{i\in[M]} r_x^i$. We will refer to this approach as the *realized reward oracle*, as it observes both the task and the realized performance of all models on the task before selection. In practice, matching the performance of this oracle is possible via simple ensemble methods, but is computationally exhaustive.

The goal in contextual routing is to spend notably less compute at inference time by avoiding querying all the models in the pool and instead learn a contextual strategy that can predict the best-performing model for a given task. In the next subsection, we formally characterize the gap between the expected reward and the realized reward oracle and show how much of this gap can be explained by contextual model selection.

5.1 The Gain of Contextual Model Selection

By Jensen, $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$ for any convex function f, therefore,

$$\max_{i\in[M]} \mathbb{E}_x\Big[\mathbb{E}_{y^i\sim\pi^i(\cdot|x)}\big[\mathcal{V}(x,y^i)\,\big|\,x\big]\Big] \leq \mathbb{E}_x\Big[\max_{i\in[M]} \mathbb{E}_{y^i\sim\pi^i(\cdot|x)}\big[\mathcal{V}(x,y^i)\mid x\big]\Big]$$

and,

$$\mathbb{E}_x\Big[\max_{i\in[M]} \mathbb{E}_{y^i\sim\pi^i(\cdot|x)}\big[\mathcal{V}(x,y^i)\mid x\big]\Big] \leq \mathbb{E}_x\Big[\mathbb{E}_{y^i\sim\pi^i(\cdot|x)}\big[\max_{i\in[M]} \mathcal{V}(x,y^i)\mid x\big]\Big]$$

As a result, the following ordering holds,

$$J^{ER} \leq J^{MS} \leq J^{RR}$$

The gap between best single model, and the oracle is,

$$J^{RR} - J^{ER} = \underbrace{J^{RR} - J^{MS}}_{\text{Gain of Verification}} + \underbrace{J^{MS} - J^{ER}}_{\text{Gain of Model Selection}}$$

The gain of the realized reward oracle over the best single model can be explained by the gain of model selection and the gain of observing the realized reward at inference. Contextual routing algorithms that observe no more than $\sigma(x)$ before model selection can not fully close the gap to the oracle, as part of that gap, $J^{RR} - J^{MS}$, comes from the realization of the best generated input. The verification gap can be closed by allowing the model to verify the performance of a subset of models at inference, similar to recent routing algorithms (Fisch et al., 2026) allow expensive estimations (querying at inference) whenever the cheap estimation appears to be unreliable.

As mentioned earlier, the benchmarks only provide a single generation for each query-response pair. Hence, the true model selection oracle can not be plotted on these experiments, and the best estimate of such an oracle is the realized reward oracle that we reported before.

6 Limitations

Capability Synthesis is vulnerable to the common weakness of embedding-based routers. Noisy embeddings that do not contain enough information for model selection can negatively affect the quality of routing. As a regression method, capability synthesis has an estimation error, which can affect the routing performance if there aren't a sufficient number of observations.

Capability synthesis requires tuning the rank d, and the regularization parameter λ. While this is common to most of routing methods, a hyperparameter-free version of Capability Synthesis remains an interesting extension.

7 Conclusion & Future Work

In this paper, we studied the problem of inference-time model selection in language models. We proposed capability synthesis that learns the capability structure of models and tasks, and uses this structure to recover the relative performance of models on any given task. We evaluated our method on routing benchmarks, and the results show that simple routers can outperform other baselines.

There are several interesting future works. Similar to the setting of (Fisch et al., 2026), one can consider an extension of capability synthesis by designing a confidence-based rule to decide whether we should query the model at inference and verify its generation or trust the prelearned estimates. This variation of Capability synthesis can potentially close the gap to the realized reward oracle.

One can consider replacing PCA with matrix factorizations that directly optimize for preserving the ranking of models, as opposed to SVD, which minimizes reconstruction error. In fact, model selection requires a more relaxed criterion than preserving the full ranking. It suffices that our estimates of model performance preserve the model with maximum performance, since the final selection accuracy doesn't take the full ranking of models into account.

Finally, capability synthesis is a well-suited framework to develop a better understanding of the model selection problem in language models from a statistical perspective. Our empirical results show that a simple regression-based method can outperform more complex trainable routers, which encourages better understanding of the underlying difficulties of this problem.

AI USE STATEMENT

In this work, we used generative AI tools for designing and providing feedback for research experiments, implementing methods and baselines, cleaning and reformatting datasets. We have not used generative AI tools for writing sections of the paper, but we have received feedback from AI to improve writing. Additionally, we used generative AI tools for organizing the tables and latex formatting and creating plots, and finding relevant literature, and generating synthetic datasets as a sanity check. We have reviewed all AI-assisted work. References and code have been manually checked by at least one authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

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

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

Teng-Ruei Chen. How much of the routing gap is real? decomposing the router-to-oracle gap into reproducible specialist advantage and single-draw label noise. *arXiv preprint arXiv:2607.03436*, 2026.

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

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

Adam Fisch, Shubhendu Trivedi, Fantine Huot, William W Cohen, Michael Kaisers, Mirella Lapata, Kate Larson, and Jacob Eisenstein. Pandora's ai model routing box: Efficient allocation with costly value estimation. *arXiv preprint arXiv:2608.20316*, 2026.

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.

Zhongzhan Huang, Guoming Ling, 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. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 3860–3887, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.208. URL `https://aclanthology.org/2025.findings-emnlp.208/`.

Ian T Jolliffe. *Principal component analysis*. Springer, 2002.

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

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Aviya Maimon, Amir David Nisan Cohen, Gal Vishne, Shauli Ravfogel, and Reut Tsarfaty. From benchmarks to skills: Low-rank factors for llm evaluation. *Transactions of the Association for Computational Linguistics*, 14:1639–1653, 2026.

Yasmin Moslem and John D Kelleher. Dynamic model routing and cascading for efficient llm inference: A survey. *arXiv preprint arXiv:2603.04445*, 2026.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5503a7c69d48a2f86fc00b3dc09de686-Paper-Conference.pdf.

Yangjun Ruan, Chris J Maddison, and Tatsunori Hashimoto. Observational scaling laws and the predictability of langauge model performance. *Advances in Neural Information Processing Systems*, 37:15841–15892, 2024.

Reza Shirkavand, Shangqian Gao, Peiran Yu, and Heng Huang. Cost-aware contrastive routing for llms. *Advances in Neural Information Processing Systems*, 38:155627–155659, 2026.

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

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *Advances in neural information processing systems*, 33: 16857–16867, 2020.

Martin Wainwright. *High-dimensional statistics: a non-asymptotic viewpoint*. Number 48 in Cambridge series in statistical and probabilistic mathematics. Cambridge University Press, Cambridge ; New York, NY, 2019. ISBN 978-1-108-49802-9.

Yi-Kai Zhang, De-Chuan Zhan, and Han-Jia Ye. Capability instruction tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 25958–25966, 2025a.

Yiqun Zhang, Hao Li, Chenxu Wang, Linyao Chen, Qiaosheng Zhang, Peng Ye, Shi Feng, Daling Wang, Zhen Wang, Xinrun Wang, et al. The avengers: A simple recipe for uniting smaller language models to challenge proprietary giants. *arXiv preprint arXiv:2505.19797*, 2025b.

Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. Embedllm: Learning compact representations of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 76913–76926, 2025.

A Appendix

A.1 Additional Algorithms

Algorithm 3: Model Pool Extension

Input: New model π^{M+1}, Reward vector $\mathbf{r}^{M+1} \in \mathbb{R}^N$, Tolerance threshold $\tau \in [0, 1]$

Center reward vector using previous mean, $\tilde{\mathbf{r}}^{M+1} = \mathbf{r}^{M+1} - \bar{\mathbf{r}}$

Estimate the capability profile of π^{M+1} in the learned capability space

$$\mathbf{p}^{M+1} = \arg\min_{p \in \mathbb{R}^d} \|\tilde{\mathbf{r}}^{M+1} - Qp\| = \Sigma_d^{-1/2} V_d^T \tilde{\mathbf{r}}^{M+1}$$

Compute Residual $\mathbf{e} = \tilde{\mathbf{r}}^{M+1} - Q\mathbf{p}^{M+1}$

if $\frac{M}{M+1}\|\mathbf{e}\|^2 \geq \tau(\sigma_d^2 - \sigma_{d+1}^2)$ **then**

  Refit Capability Synthesis

else

  Add π^{M+1} to the pool with previous synthesizer $\mathbf{q}(x)$ and capability profiles $\mathbf{p}^{M+1} \cup \{\mathbf{p}^i\}_{i=1}^M$

Table 2: EmbedLLM test Pass@1 (%) on strong-to-weak pools. Each column is the pool size.

Router	5	10	25	50	112
Oracle	77.2	80.8	89.6	93.7	98.0
Capability Synthesis	63.5	63.8	**64.3**	**64.3**	**66.6**
Avengers	**63.6**	**63.9**	63.7	63.7	66.0
EmbedLLM MF	**63.6**	63.7	**64.3**	64.0	66.1
GraphRouter	60.3	60.5	60.6	60.7	60.8
MLP	60.7	60.8	60.7	60.7	62.1
Best single model	60.7	60.7	60.7	60.7	60.7
Weighted random	58.8	58.5	57.1	56.3	52.9
Random	57.4	56.0	52.0	48.7	40.7

Remarks

1. Centering $\mathbf{r}^{M+1}$ with the previous mean $\bar{\mathbf{r}}$ doesn't affect selection, as the selection strategy 2 is invariant to the offset vector $\mathbf{b} \in \mathbb{R}^N$ that is common to all models. If every profile is computed as $\Sigma_d^{-1/2} V_d^T(\mathbf{r}^i - \mathbf{b})$, then $\hat{i}_x$ does not depend on $\mathbf{b}$.
2. The refit criteria $\frac{M}{M+1}\|\mathbf{e}\|^2 \geq \tau(\sigma_d^2 - \sigma_{d+1}^2)$ captures whether the residual is large enough to add a new capability direction. Note that this is a first-order heuristic criterion and can be replaced by statistical tests or standard PCA criteria with provable guarantees (Jolliffe, 2002; Wainwright, 2019).

A.2 Experiments

A.2.1 Benchmark Details

- **RouterEval** (Huang et al., 2025) contains over 200 million model-task performance record of the evaluation of LLMs on different benchmarks. Each benchmark is split into train, test, eval with ratio (8, 10, 10). For each benchmark, they construct candidate pools of size $M \in \{3, 5\}$ (easy pools), $M \in \{10, 100, 1000\}$ (hard pools). There are 12 benchmarks used including ARC, HellaSwag, MMLU, MMLU-Pro, TruthfulQA, WinoGrande, GSM8k, IFEVAL, BBH, GPQA, MuSR, MATH Lvl 5.
- **EmbedLLM** (Zhuang et al., 2025) contains the performance trace of 112 open-source LLMs on 36054 tasks in the form of a dense binary reward matrix, which we use as our performance matrix R. Questions are split into train, test, eval with ratio (8, 10, 10). There are 10 benchmarks used including MMLU, TruthfulQA, SocialQA, PIQA, MedMCQA, MathQA, LogiQA, GSM8k, GPQA, ASDiv.

A.2.2 Routing Benchmark Results

We report the numerical values for pass@1, and regret to oracle for both EmbedLLM and RouterEval Benchmarks below.

A.2.3 Baselines Configurations

Table 3: EmbedLLM test regret versus the oracle on strong-to-weak pools.

Router	5	10	25	50	112
Oracle	0.000	0.000	0.000	0.000	0.000
Capability Synthesis	0.137	0.170	**0.253**	**0.293**	**0.314**
Avengers	0.136	**0.169**	0.259	0.300	0.320
EmbedLLM MF	**0.135**	0.171	**0.253**	0.297	0.319
GraphRouter	0.168	0.203	0.289	0.329	0.372
MLP	0.164	0.200	0.288	0.329	0.358
Best single model	0.164	0.201	0.288	0.329	0.372
Weighted random	0.183	0.223	0.324	0.374	0.450
Random	0.197	0.248	0.376	0.449	0.573

Table 4: RouterEval test Pass@1 (%), averaged over datasets and pool groups. Each column is the pool size.

Router	5	10	100	1000
Oracle	83.7	89.0	96.4	98.6
Capability Synthesis	61.0	**59.8**	**60.5**	**66.2**
Avengers	**61.1**	59.5	59.3	64.3
EmbedLLM MF	58.0	57.2	56.1	61.9
GraphRouter	58.8	57.6	56.5	35.1
MLP	48.2	45.4	41.0	35.4
Best single model	59.3	57.4	55.7	62.9
Weighted random	59.3	58.0	56.7	62.1
Random	51.6	49.8	47.7	46.3

Table 5: RouterEval test regret versus the oracle, averaged over datasets and pool groups.

Router	5	10	100	1000
Oracle	0.000	0.000	0.000	0.000
Capability Synthesis	0.227	**0.292**	**0.359**	**0.324**
Avengers	**0.226**	0.295	0.371	0.343
EmbedLLM MF	0.258	0.319	0.402	0.366
GraphRouter	0.249	0.314	0.399	0.635
MLP	0.356	0.436	0.554	0.631
Best single model	0.245	0.316	0.407	0.357
Weighted random	0.245	0.311	0.397	0.364
Random	0.321	0.392	0.487	0.522

Table 6: Hyperparameters for EmbedLLM and RouterEval. Rows marked by a star mean that the value was selected on the validation split of the datasets.

Method	Hyperparameter	Value
Capability synthesis	Factorization	Truncated SVD, sign-aligned
	Synthesizer	Closed-form ridge
	Rank d	$\{1, 2, 4, 8, 16, 32, 64\}^\star$
	Ridge λ	$\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2\}^\star$
EmbedLLM MF	Model embedding dim	232
	Embedding noise α	0.05
	Classifier	Linear, 2-way
	Optimizer	Adam
	Learning rate	10^{-4}
	Weight decay	10^{-5}
	Batch size / epochs	2048 / 50
	Early stopping	Patience 5 on validation reward
GraphRouter	Hidden / LLM dim	8 / 64
	Layers	2 message-passing layers
	Optimizer	AdamW
	Learning rate	3×10^{-4}
	Weight decay	10^{-4}
	Train edge mask	0.5
	Epochs / patience	50 / 8
	Loss	Weighted BCE, positive weight $M - 1$
Avengers	Clusters k	$\{8, 16, 32, 64, 128\}^\star$
	Samples per query	1
	Clustering	Mini-batch k-means, 10 inits
MLP classifier	Hidden layer	128, ReLU
	Output	Softmax over M models
	Optimizer	Adam, $\ell_2 = 10^{-4}$, lr 10^{-3}
	Max iterations	200
	Early stopping	10% of the training queries