Inference-Time Model Selection via Capability Synthesis

Aida Afshar[1], Siddharth Mishra-Sharma[1,2], and Aldo Pacchiano[1,3]

[1]Boston University
[2]Anthropic
[3]Broad Institute of MIT and Harvard

November 2026

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 given 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. 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

Benchmarks and model evaluations record how a large collection of language models perform on tasks of varying difficulty. These performance records provide information about the joint space of models and tasks, and reveal shared patterns of capabilities among models [7, 8]. This creates an opportunity to study the structure behind this joint space, and develop methods to understand the landscape of the capabilities of language models and their downstream applications [14, 18].

In this paper, we study the problem of inference-time model selection, where we have access to a finite pool of models at inference. This pool can consist of models with different scale, alignment, architecture, or a single model paired with different inference-time harnesses or sampling strategies. The goal in model selection is to design an algorithm that can select the best model for a given task, without exhaustively querying all the models at inference. This is also known as the routing problem, and several papers have studied this problem in the literature of language models [20, 1, 26, 25, 15].

Our approach to this problem is to design a router that learns a shared structure of capabilities where both models and tasks can be represented, and use this structure to guide model selection. We propose a simple framework, named Capability Synthesis, that uses the performance matrix of the models on a set of calibration 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

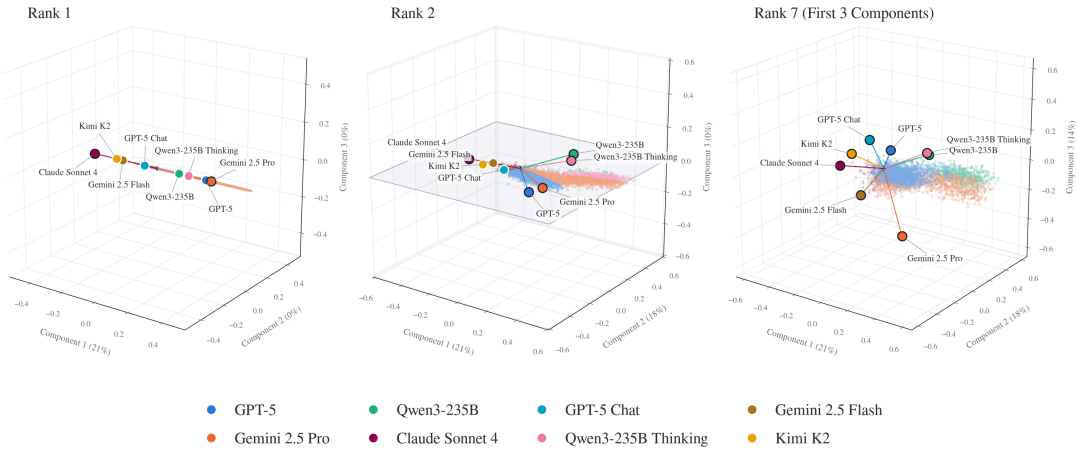


Figure 1: Capability Synthesis jointly represents models and tasks in a shared capability space and leverages this structure to guide inference-time model selection. Here, models and tasks are represented through capability profiles and task specifications, and tasks are colored according to the model that was selected for them by Capability Synthesis.

receive at inference. Additionally, we show a simple procedure to extend the pool of models with minimal refitting cost.

We evaluate Capability Synthesis on large-scale routing benchmarks [8, 26] and compare it to several existing routing strategies and model selection baselines. Our empirical results show that simple regularized regression methods can compete with and outperform more complex routers, particularly larger pools of models.

In addition to contextual model selection, we consider two different deployment strategies, i.e., single-model deployments and ensemble methods. For each deployment strategy, we formally define an oracle comparator based on the information they can observe before selection. We analyze the gap between the best single-model deployment and an oracle that can verify the generation of every model before selection, previously studied by [3]. In a simple argument that holds for any verifier, we show that part of the gap to the oracle can be explained by contextual model selection, and the remaining part is the gain of verification at inference. We summarize our contributions as follows:

- We propose Capability Synthesis that learns a closed-form representation of models and tasks in a shared capability space, and design a contextual model selection strategy that uses these representations for inference-time model selection in language models.
- Our experiments on two comprehensive routing benchmarks show that Capability Synthesis can closely match or outperform state-of-the-art routing and model selection strategies, with particularly good performance on larger pools of models.
- We conduct an ablation study on the role of PCA and show how spectral regularization can improve downstream model selection performance. We study both cases where retaining few principal components is effective for model selection, and regimes where capability synthesis recovers the solution of direct ridge regression at full rank.

2 Related Work

Prior literature has studied various algorithms and architectures for routing and model selection in language models. EmbedLLM [26] uses an encoder-decoder architecture to learn general-purpose embeddings for models and tasks, which can be utilized for benchmark performance prediction and routing. The encoder learns two separate embedding models, a model embedding network and a question embedding network, and the decoder is a linear classifier that uses the embeddings to outputs a binary label that predicts whether the model can solve the question or not. They train the parameters with binary cross-entropy loss using gradient-based learning algorithms.

Other routers use contrastive methods for routing and model selection. RouterDC [2] proposes a contrastive learning method to address a common failure of routers when multiple LLMS succeed on queries. Cost-aware contrastive routing [19] learns a shared representation of models and tasks via a cost-aware contrastive objective and trains and selects models using nearest neighbor search.

Early work of [20] formulates routing as a collection of binary classification problems. Several methods study routing for cascades such as HybridLLM [5] and FrugalGPT [1]. RouteLLM [16] uses human preference data to dynamically select between a strong and weak LLM during inference to balance performance and cost. Model-SAT router [24] learns representations through instruction tuning, and uses an aptitude test for adding new models to the pool. The Avengers [25] clusters queries in a pretrained embedding space and route each query according to cluster-level model performance.

Apart from routing literature, estimating capability directions using PCA on benchmark matrices has been studied by [18], where they used these results to derive scaling laws for language models and predict emergent capabilities with scale. The Big-bench study [22] systematically evaluates a large collection of language models, demonstrating different capabilities and scaling behaviour accross tasks. More recently, [14] study latent skill factors in benchmark evaluations to characterize model capabilities. Our work builds on this perspective by using observed model performances not only to characterize model capabilities, but also task-specific capability representations for inference-time model selection.

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.

Algorithm 1: Capability Synthesis

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

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`

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

$$\tilde{\mathbf{R}} \approx \mathbf{U}_d\mathbf{\Sigma}_d\mathbf{V}_d^T = \left(\mathbf{U}_d\sqrt{\mathbf{\Sigma}_d}\right)\left(\mathbf{V}_d\sqrt{\mathbf{\Sigma}_d}\right)^T = \mathbf{P}\mathbf{Q}^T$$

`// Phase 2. Ridge Regression`

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

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 $\hat{i}_x$

Receive query x,

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

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

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

return y_x, $\hat{i}_x$

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 $\{v_k\}_{k=1}^d \subseteq \mathbb{R}^N$, where $1 \leq d \leq \min(M, N)$, and $v_k \in \mathbb{R}^N$ is the k'th direction that maximizes the average pairwise model separation,

$$\mathbf{v}_k = \underset{\substack{\|\mathbf{v}\|=1 \\ \mathbf{v} \perp \langle v_1, \ldots, v_{k-1} \rangle}}{\arg\max} \frac{1}{2M^2} \sum_{i=1}^M \sum_{l=1}^M \left[(\mathbf{r}^i - \mathbf{r}^l)^T\mathbf{v}\right]^2$$

where $\mathbf{r}^i \in \mathbb{R}^N$ is the i'th row of the performance matrix R. The unit vectors $\{v_k\}_{k=1}^d$ are d capability directions along which the models' capabilities are most distinguishable. We will refer to the d-dimensional space spanned by these directions $\langle v_1, \ldots, v_d \rangle \subseteq \mathbb{R}^N$ 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 i'th element $\mathbf{p}^i_k$ 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 $\mathbf{R} \in \mathbb{R}^{M \times N}$. Similar to the standard PCA procedure, we first subtract the column mean from each element of that column,

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

and then perform Singular Value Decomposition (SVD) on the column-centered matrix $\tilde{\mathbf{R}}$,

$$\tilde{\mathbf{R}} \approx \mathbf{U}_d\mathbf{\Sigma}_d\mathbf{V}_d^T = \left(\mathbf{U}_d\sqrt{\mathbf{\Sigma}_d}\right)\left(\mathbf{V}_d\sqrt{\mathbf{\Sigma}_d}\right)^T = \mathbf{P}\mathbf{Q}^T$$

where,

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

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

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

where $\tilde{\mathbf{r}}^i = \mathbf{r}^i - \bar{\mathbf{r}}$ and $\bar{\mathbf{r}} = \frac{1}{M}\sum_{i=1}^{M}\mathbf{r}^i$. The value $(\mathbf{r}^i - \mathbf{r}^l)^T\mathbf{v}$ captures the pairwise differences between the performance of π^i and π^l towards direction $\mathbf{v}$. Therefore, $\{\mathbf{v}_k\}_{k=1}^d$ discovered by PCA correspond to directions towards which the average pairwise model separation is maximized. We will use $\mathbf{p^i} \in \mathbb{R}^d, i \in [M]$ as the capability profile of model $\pi^i \in \Pi$, and $\mathbf{q}_j \in \mathbb{R}^d, j \in [N]$ as the synthetic 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 $\mathcal{D}_{cal} = \{(\mathbf{x}_j, \mathbf{q}_j)\}_{j=1}^N$, and learn synthesizer parameters $\theta \in \mathbb{R}^{D \times d}$ via Ridge Regression,

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

where $\mathbf{X} \subseteq \mathbb{R}^{N \times D}$ is the query embeddings $\{\mathbf{x}_j\}_{j=1}^N$ 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 $\mathbf{q}(x) = \hat{\theta}^T \mathbf{x}$ 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.

Remarks In the PCA phase of capability synthesis, we choose to equally split the covariance matrix $\mathbf{\Sigma}_d$ between $\mathbf{P}$ and $\mathbf{Q}$, so that both the capability profiles $\{\mathbf{p}^i\}_{i=1}^M$, and task specifications $\{\mathbf{q}_j\}_{j=1}^N$ encode the singular values. Note that, this decomposition is not unique, since $\forall a \in (0, 1)$,

$$\mathbf{P} = \mathbf{U}_d \mathbf{\Sigma}_d^a, \ \mathbf{Q} = \mathbf{V}_d \mathbf{\Sigma}_d^{1-a},$$

is a valid decompositions. Despite this fact, the final selected index is invariant to this transformation. At inference time, the model selection strategy is designed to be the dot product $\hat{i}_x = \arg\max_{i \in [M]} \mathbf{q}^T(x) \mathbf{p}^i$, which remains the same for any split $a \in (0, 1)$.

3.3 Model Pool Extension

Consider a setup where Capability Synthesis has been fitted on a performance matrix $\mathbf{R} \in \mathbb{R}^{M \times N}$ 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,

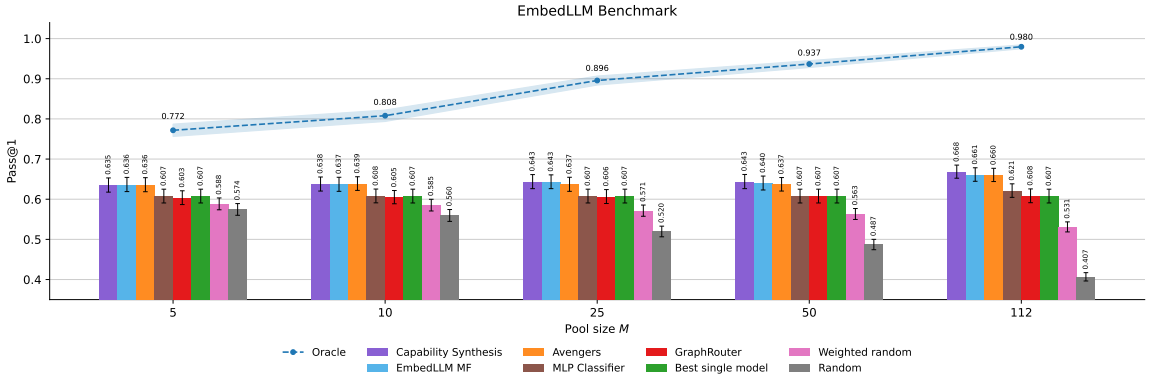
$$\mathbf{p}^{M+1} = \arg\min_{\mathbf{p} \in \mathbb{R}^d} \|\tilde{\mathbf{r}}^{M+1} - \mathbf{Q}\mathbf{p}\| = \mathbf{\Sigma}_d^{-1/2} \mathbf{V}_d^T \tilde{\mathbf{r}}^{M+1}$$

We then 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 B.

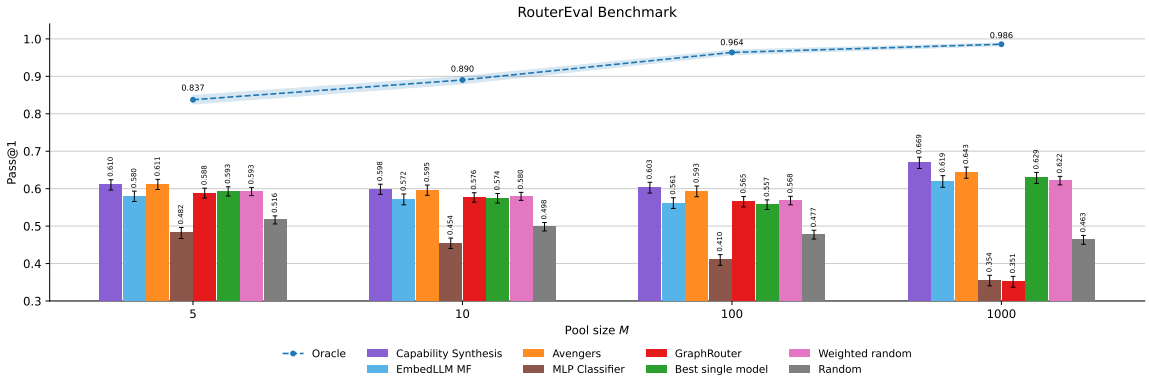
4 Experiments

4.1 Routing Benchmarks

We evaluate the performance of capability synthesis on two comprehensive routing benchmarks. The first one is **RouterEval** [8] contains over 200 million model-task performance record of the evaluation of LLMs on 12 benchmarks. For each benchmark, they construct two types of candidate pools (easy pools, and hard pools) to test the performance of routers on different types of pools. The second one is **EmbedLLM** [26] 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.



(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 2: 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.

Baselines We compare capability synthesis with eight routing baselines: Avengers [25] which is a clustering-based method; GraphRouter [6] 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 [26]. We train a simple MLP Classifier with one hidden layer and ReLU activation that learns to predict 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 [26]. The weighted 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 [21] for EmbedLLM, and RoBERTa (768-dim) [13] for RouterEval. For EmbedLLM, we use the strong-to-weak 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 C.3.

Results Figure 2 summarizes the results. We report pass@1 versus the size of the model pool on RouterEval and EmbedLLM benchmarks. We observe that Capability Synthesis, Avengers, and

Table 1: RouterEval test Pass@1 for three model pool compositions: All-Strong, All-Weak, Strong-to-Weak model pools. Each cell shows the mean ± 95% bootstrap interval.

	All-strong				All-weak				Strong-to-weak			
Router / M	5	10	100	1000	5	10	100	1000	5	10	100	1000
Oracle	90.7 ± 1.4	93.2 ± 1.2	97.2 ± 0.7	98.9 ± 0.4	72.3 ± 2.1	81.3 ± 1.9	94.0 ± 1.1	97.4 ± 0.7	88.2 ± 1.5	92.5 ± 1.1	98.0 ± 0.6	99.3 ± 0.2
Capability Synthesis	**76.1 ± 1.9**	**73.6 ± 1.9**	**72.3 ± 2.1**	**78.1 ± 1.8**	34.8 ± 2.2	**34.0 ± 2.2**	33.7 ± 2.1	**45.0 ± 2.4**	72.2 ± 1.9	72.0 ± 1.8	**74.8 ± 1.8**	**77.8 ± 1.8**
EmbedLLM MF	73.7 ± 2.0	70.9 ± 2.1	69.1 ± 2.0	75.7 ± 2.0	33.9 ± 2.2	33.1 ± 2.2	32.4 ± 2.2	38.6 ± 2.2	66.3 ± 2.1	67.5 ± 2.0	67.0 ± 2.1	71.5 ± 2.1
Avengers	74.8 ± 2.0	73.4 ± 2.0	71.7 ± 2.0	75.9 ± 1.9	**35.5 ± 2.2**	33.8 ± 2.2	**34.5 ± 2.2**	40.1 ± 2.2	**73.0 ± 1.9**	71.4 ± 1.9	71.6 ± 2.0	76.8 ± 1.8
MLP Classifier	70.5 ± 1.9	69.6 ± 2.0	63.8 ± 2.1	55.0 ± 2.1	33.5 ± 2.2	31.9 ± 2.1	31.7 ± 2.3	23.1 ± 1.9	40.5 ± 2.1	34.8 ± 1.9	27.4 ± 2.0	28.2 ± 1.9
GraphRouter	73.4 ± 1.9	70.5 ± 1.9	67.7 ± 1.9	57.7 ± 2.2	30.2 ± 2.1	29.5 ± 2.0	29.5 ± 2.1	23.9 ± 2.0	72.9 ± 1.9	**72.9 ± 1.8**	72.3 ± 2.0	23.8 ± 1.9
Best single model	75.1 ± 1.9	70.1 ± 2.0	67.0 ± 2.0	76.1 ± 1.8	29.8 ± 2.1	29.3 ± 2.0	27.9 ± 1.9	36.5 ± 2.1	72.9 ± 1.9	**72.9 ± 1.8**	72.3 ± 2.0	76.1 ± 1.9
Weighted random	73.3 ± 1.4	70.9 ± 1.4	68.4 ± 1.5	74.6 ± 1.4	32.9 ± 1.4	31.8 ± 1.4	31.6 ± 1.3	38.1 ± 1.3	71.5 ± 1.6	71.2 ± 1.5	70.5 ± 1.5	73.7 ± 1.4
Random	71.9 ± 1.4	69.7 ± 1.4	64.5 ± 1.4	59.7 ± 1.5	33.1 ± 1.3	32.1 ± 1.2	31.5 ± 1.2	31.4 ± 1.2	49.9 ± 1.3	47.8 ± 1.3	47.2 ± 1.3	48.0 ± 1.3

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. Detailed numerical results are reported Tables 5 and 6.

Additionally, the RouterEval dataset consists of three pool types with different model compositions: All-weak models, All-strong models, Strong-to-weak models, where we report the average score across these pool types. To better understand the performance of Capability Synthesis and baselines on different pool compositions, we provide test pass@1 for each individual pool type in table 1. In these results, Capability Synthesis achieves highest mean pass@1 in eight of twelve settings, and leads accross all three pool types at $M = 1000$.

Increasing the size of the pool adds more complexity to routing, but provides more observations to better estimate the capability profiles $\mathbf{p}^i, i \in [M]$ and task specifications $\mathbf{q}_j, j \in [N]$ in Capability Synthesis. Our empirical results also confirm the remaining gap to oracle discussed by recent literature on routing [11, 3], which we thoroughly break down in section 5. Detailed numerical results for pass@1 and regret of each baseline and oracles are available in Appendix C.1.1.

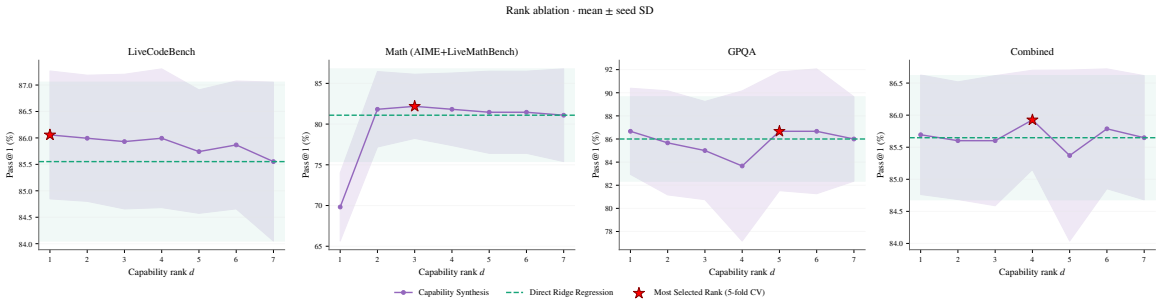
4.2 The Role of Truncation

In this section we study the role of truncation in Capability Synthesis and conduct an ablation on the PCA phase of the algorithm. Importantly, we study when truncation provides an advantage to capability synthesis over direct ridge regression.

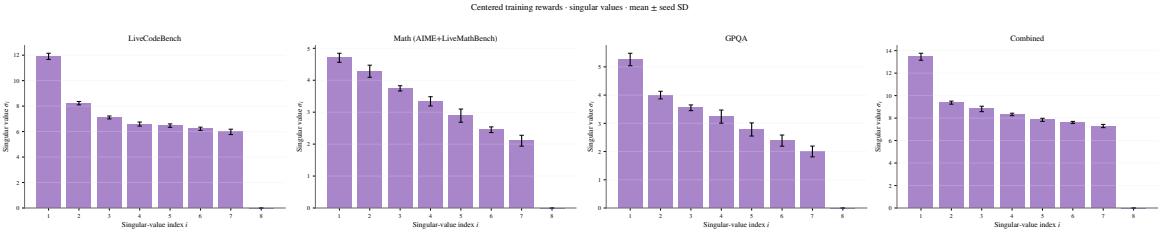
First, we formally show that given performance matrix $\mathbf{R} \in \mathbb{R}^{M \times N}$ and regularization parameter $\lambda \in \mathbb{R}$, Capability Synthesis at full rank recovers the solution of direct ridge regression (Theorem A.1 in Appendix A). Therefore, the solution of direct ridge regression is always withing the candidate solutions of capability synthesis.

Second, we evaluate Capability Synthesis on a pool of eight models and three specialized domains, conduct a rank ablation, and provide a side-by-side comparison to direct ridge regression. The models include Claude Sonnet 4, Gemini 2.5 Flash, Gemini 2.5 Pro, GPT-5 Chat, GPT-5, Qwen3-235B-A22B-2507, Qwen3-235B-A22B-Thinking-2507, and Kimi K2-0905. The domains include coding, math, scientific reasoning, and the aggregated dataset of these three domains, where we used LiveCodeBench [9] for coding, AIME [4] and LiveMathBench [12] for Math, and GPQA [17] for scientific reasoning.

In Figure 3-a, we report the performance of Capability Synthesis versus rank d, and compare



(a) Test Pass@1 vs rank. Capability Synthesis outperforms direct ridge regression at lower ranks, leveraging the spectral regularization provided by PCA. Capability Synthesis recovers direct ridge regression at full rank.



(b) PCA of the performance matrix $\mathbf{R}$ across three individual datasets and the aggregated data. The columns are singular values of the column-centered performance matrix.

Figure 3: Capability Synthesis on a pool of eight models, on three individual domains: Coding (LiveCodeBench), Math (AIME+LiveMathBench), Scientific Reasoning (GPQA), and the aggregated dataset of all domains. Despite weak evidence of low-rank structure, a few principal directions contain the information required for model selection

it to direct ridge regression. The results show that retaining only some of the components can improve on direct Ridge Regression, that is matched at full rank. Figure 3-b shows the singular values of the performance matrix of each domain and the combined dataset. Here, singular values are gradually decaying, with substantial variation remaining beyond the first few components. Despite little evidence of a sharply low-rank performance matrix, truncation improves downstream model selection performance on these domains. Hence, the objective of model selection, which is identifying $i_x^\star = \arg\max_{i \in [M]} r_x^i$, can require fewer principal dimensions than accurately reconstructing the entire performance matrix.

The effect of truncation varies across domains and model pool compositions. On LiveCodeBench, rank 1 achieves the best performance and outperforms Capability Synthesis at higher ranks, whereas in Math, the first three components are required to get the best model selection accuracy. In certain regimes, where preserving all principal components are required to reliably learn capability profiles and task specification, capability synthesis achieves its best performance at full rank, recovering the solution of direct ridge regression.

We conducted this ablation on twelve individual datasets of RouterEval and ten datasets of EmbedLLM, and compared Capability Synthesis with direct ridge regression. The plots are available in Appendix C.2. We similarly observe that the effect of truncation varies across domains. On specialized domains, truncation often helps Capability Synthesis outperform or match direct ridge regression at lower rank, whereas in combined domains, performance generally increases with preserving more directions. Particularly, on combined dataset of EmbedLLM, Capability Synthesis achieves its best performance at full rank, showing that PCA doesn't contribute to that baseline.

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

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 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.

5.1 The Gain of Contextual Model Selection

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

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

and,

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

As a result, the following ordering holds for any verifier $\mathcal{V} : \mathcal{X} \times \mathcal{Y} \to [0, 1]$,

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

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 output. The verification gap can be closed by allowing the model to verify the performance of a subset of models at inference [3]. Both the benchmarks provide a single generation for each query-response pair. Hence, the true model selection oracle can not be plotted on these experiments. The model selection oracle requires $\max_{i \in [M]} \mathbb{E}_{y^i \sim \pi^i(\cdot|x)} \left[\mathcal{V}(x, y^i) \mid x \right]$, which requires multiple generations of model π^i for each input x to estimate the expectation. Therefore, we've only compared Capability Synthesis and other baselines with the expected reward oracle (best single deployment), and the realized reward oracle.

6 Limitations

Capability Synthesis requires tuning the rank d, and the regularization parameter λ. While hyperparameter tuning is common to many routing methods, a hyperparameter-free version of Capability Synthesis remains an interesting extension. Additionally, task embeddings that do not contain enough information for model selection can negatively affect the quality of routing in Capability Synthesis and other embedding-based routers.

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. There are several interesting future works. 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. 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.

References

[1] 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.

[2] 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.

[3] 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.

[4] Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL https://arxiv.org/abs/2605.00674.

[5] 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, pages 41348–41366, 2024.

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

[7] 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.

[8] 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, editors, *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 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/.

[9] 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.

[10] Ian T Jolliffe. *Principal component analysis*. Springer, 2002.

[11] 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*, pages 37733–37754, 2026.

[12] Junnan Liu, Hongwei Liu, Linchen Xiao, Ziyi Wang, Kuikun Liu, Songyang Gao, Wenwei Zhang, Songyang Zhang, and Kai Chen. Are your llms capable of stable reasoning? *arXiv preprint arXiv:2412.13147*, 2024.

[13] 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.

[14] 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.

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

[16] 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, editors, *International Conference on Learning Representations*, volume 2025, pages 34433–34448, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5503a7c69d48a2f86fc00b3dc09de686-Paper-Conference.pdf.

[17] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

[23] 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.

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

[25] 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*, 2025.

[26] 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, pages 76913–76926, 2025.

Appendix

A Analysis

Capability Synthesis at full rank provably recovers direct ridge regression on the performance matrix $\mathbf{R}$. Hence, the solution of Ridge regression is always contained within the candidate solutions of capability synthesis. The role of the PCA phase is to allow spectral regularization, and shows how each principal component contribute to routing. In the regimes where spectral regularization is helpful, capability synthesis can outperform direct ridge regression at lower rank. Whenever truncation hurts performance, capability synthesis keeps all the principle components and recovers direct ridge regression. We formalize this statement below, and refer to ablations 4.2 for empirical results.

Theorem A.1. *Let $\mathbf{R} \in \mathbb{R}^{M \times N}$ be the performance matrix of pool $\Pi = \{\pi^i\}_{i=1}^M$ on calibration tasks $\{\mathbf{x}_j\}_{j=1}^N$, , and let $\mathbf{X} \in \mathbb{R}^{N \times D}$ be the corresponding task embedding matrix. For any $\lambda > 0$ task $x \in \mathcal{X}$, the selected model by Capability Synthesis at full rank is equal to the selected model by direct ridge regression on $\mathbf{R}$.*

Proof. Direct ridge regression on $\mathbf{R}$ solves the following minimization problem,

$$\bar{\theta} = \arg\min_{\theta} \|\mathbf{X}\theta - \mathbf{R}^{\mathbf{T}}\|_F^2 + \lambda\|\theta\|_F^2$$

For task $x \in \mathcal{X}$, it predicts,

$$\bar{s}_x = \bar{\theta}^\top \phi(x) = \mathbf{R}\mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\phi(x) \in \mathbb{R}^M. \tag{2}$$

Capability Synthesis first centers the performance matrix and computes its rank-d approximation,

$$\widetilde{\mathbf{R}} = \left(\mathbf{I}_M - \frac{1}{M}\mathbf{1}\mathbf{1}^\top\right)\mathbf{R}, \qquad \widetilde{\mathbf{R}}_d = \mathbf{P}\mathbf{Q}^\top, \tag{3}$$

where $\mathbf{P} \in \mathbb{R}^{M \times d}$ and $\mathbf{Q} \in \mathbb{R}^{N \times d}$. It then solves

$$\widehat{\theta} = \arg\min_{\theta \in \mathbb{R}^{D \times d}} \|\mathbf{X}\theta - \mathbf{Q}\|_F^2 + \lambda\|\theta\|_F^2. \tag{4}$$

The predicted model performance vector is therefore

$$\begin{aligned}
\widehat{s}_x &= \mathbf{P}\widehat{\theta}^\top\phi(x) && (5)\\
&= \mathbf{P}\mathbf{Q}^\top\mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\phi(x) && (6)\\
&= \widetilde{\mathbf{R}}_d\mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\phi(x). && (7)
\end{aligned}$$

At full rank, $d = \text{rank}(\widetilde{\mathbf{R}})$, so $\widetilde{\mathbf{R}}_d = \widetilde{\mathbf{R}}$. Therefore,

$$\begin{aligned}
\widehat{s}_x &= \left(\mathbf{I}_M - \frac{1}{M}\mathbf{1}\mathbf{1}^\top\right)\mathbf{R}\mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\phi(x) && (8)\\
&= \left(\mathbf{I}_M - \frac{1}{M}\mathbf{1}\mathbf{1}^\top\right)\bar{s}_x && (9)\\
&= \bar{s}_x - b_x\mathbf{1}, && (10)
\end{aligned}$$

where,

$$b_x = \frac{1}{M}\sum_{i=1}^{M}\bar{s}_x^i \tag{11}$$

is a scalar independent of the model index i. Hence,

$$\arg\max_{i\in[M]} \hat{s}_x^i = \arg\max_{i\in[M]} (\bar{s}_x^i - b_x) = \arg\max_{i\in[M]} \bar{s}_x^i. \tag{12}$$

The second equality holds because b_x is common across all $i \in [M]$. Finally,

$$\hat{i}_x = \arg\max_{i\in[M]} \hat{s}_x^i = \arg\max_{i\in[M]} \bar{s}_x^i = \bar{i}_x$$

B 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 $\mathbf{p}^{M+1} \cup \{\mathbf{p}^i\}_{i=1}^M$

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 [10, 23].

C Experiments

C.1 Benchmark Details

- **RouterEval** [8] 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 of roughly (80, 10, 10) that varies by dataset. We used the splits that were suggested by the benchmark. In total, there are 12 datasets used including ARC, HellaSwag, MMLU, MMLU-Pro, TruthfulQA, WinoGrande, GSM8k, IFEVAL, BBH, GPQA, MuSR, MATH Lvl 5.
- **EmbedLLM** [26] 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 (83.18 / 8.41 / 8.41) suggested by the benchmark. There are 10 benchmarks used including MMLU, TruthfulQA, SocialQA, PIQA, MedMCQA, MathQA, LogiQA, GSM8k, GPQA, ASDiv.

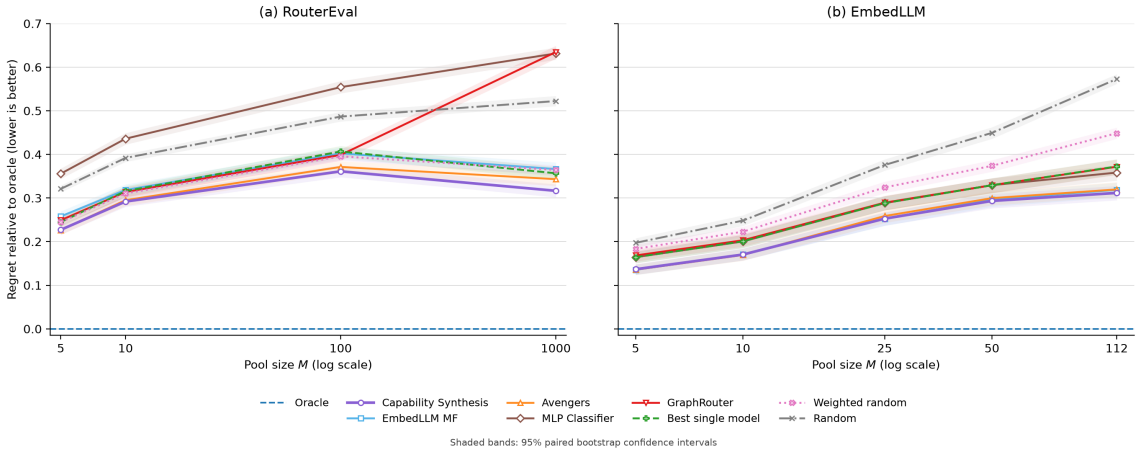


Figure 4: RouterEval and EmbedLLM test regret relative to the realized reward oracle vs. Size of the Model Pool M

Table 3: RouterEval test regret relative to the realized reward oracle (lower is better). Values are mean ± half the width of the 95% bootstrap interval, within each dataset, averaging the 12 datasets and three pool groups equally.

Router / Pool size M	5	10	100	1000
Capability Synthesis	0.227 ± 0.011	**0.292 ± 0.012**	**0.361 ± 0.013**	**0.316 ± 0.015**
EmbedLLM MF	0.258 ± 0.012	0.319 ± 0.012	0.402 ± 0.014	0.366 ± 0.015
Avengers	**0.226 ± 0.011**	0.295 ± 0.012	0.371 ± 0.013	0.343 ± 0.015
MLP Classifier	0.356 ± 0.012	0.436 ± 0.013	0.554 ± 0.014	0.631 ± 0.014
GraphRouter	0.249 ± 0.011	0.314 ± 0.012	0.399 ± 0.013	0.635 ± 0.015
Best single model	0.245 ± 0.010	0.316 ± 0.012	0.407 ± 0.013	0.357 ± 0.014
Weighted random	0.245 ± 0.008	0.311 ± 0.009	0.396 ± 0.010	0.364 ± 0.011
Random	0.321 ± 0.008	0.392 ± 0.009	0.487 ± 0.011	0.522 ± 0.012

Additionally, we used the publicly available dataset of LLMRouterBench (Cost pool) [11] and extracted performance records of the pool of eight models for plotting the shared capability structure in Figure 1, as well as individual domains of LiveCodeBench, LiveMathBench+AIME, and GPQA 3.

C.1.1 Routing Benchmark Results

We report the numerical values for Pass@1, and regret to the realized reward oracle for both EmbedLLM and RouterEval Benchmarks below. For each router-pool size score, we report a confidence interval, computed via bootstrapping on test split of the data.

For every test data x_j on router l, we consider the realzied reward $r_j^{(l)}$ of the model that router l selected. The oracle's reward is the per-question maximum over the pool, the best single model's is the reward of the model with the highest training mean, random is the per-question mean over the pool, and weighted random is the mean weighted by the empirical selection frequencies of Capability Synthesis. Pass@1 is the mean of $r_j^{(l)}$ over the test data and regret is the mean of $r_j^\star - r_j^{(l)}$, where $r_j^\star$ is the reward of oracle on x_j. For confidence intervals, we draw $B = 2000$ bootstrap resamples of test questions and recompute every quantity on each resample, with replacement, following standard bootstrap procedure.

Table 4: EmbedLLM test regret relative to the realized reward oracle (lower is better). Values are mean ± half the width of the 95% bootstrap interval, over pooled test questions.

Router / Pool size M	5	10	25	50	112
Capability Synthesis	0.137 ± 0.012	0.170 ± 0.013	**0.253 ± 0.015**	**0.293 ± 0.016**	**0.312 ± 0.016**
EmbedLLM MF	**0.135 ± 0.012**	0.171 ± 0.014	**0.253 ± 0.015**	0.297 ± 0.016	0.319 ± 0.017
Avengers	0.136 ± 0.012	**0.169 ± 0.014**	0.259 ± 0.015	0.300 ± 0.016	0.320 ± 0.017
MLP Classifier	0.164 ± 0.013	0.200 ± 0.015	0.288 ± 0.016	0.329 ± 0.017	0.358 ± 0.017
GraphRouter	0.168 ± 0.014	0.203 ± 0.015	0.289 ± 0.017	0.329 ± 0.017	0.372 ± 0.018
Best single model	0.164 ± 0.013	0.201 ± 0.015	0.288 ± 0.016	0.329 ± 0.017	0.372 ± 0.018
Weighted random	0.183 ± 0.010	0.223 ± 0.012	0.324 ± 0.013	0.374 ± 0.013	0.449 ± 0.013
Random	0.197 ± 0.011	0.248 ± 0.012	0.376 ± 0.013	0.449 ± 0.013	0.573 ± 0.011

Table 5: RouterEval test Pass@1 with 95% bootstrap confidence intervals, averaged over datasets and pool groups.

Router / Pool size M	5	10	100	1000
Oracle	83.7 ± 1.1	89.0 ± 1.0	96.4 ± 0.6	98.6 ± 0.3
Capability Synthesis (ours)	61.0 ± 1.4	**59.8 ± 1.4**	**60.3 ± 1.4**	**66.9 ± 1.5**
EmbedLLM MF	58.0 ± 1.4	57.2 ± 1.4	56.1 ± 1.4	61.9 ± 1.6
Avengers	**61.1 ± 1.4**	59.5 ± 1.4	59.3 ± 1.4	64.3 ± 1.5
MLP Classifier	48.2 ± 1.5	45.4 ± 1.4	41.0 ± 1.4	35.4 ± 1.4
GraphRouter	58.8 ± 1.3	57.6 ± 1.3	56.5 ± 1.4	35.1 ± 1.5
Best single model	59.3 ± 1.2	57.4 ± 1.3	55.7 ± 1.3	62.9 ± 1.5
Weighted random	59.3 ± 1.1	58.0 ± 1.1	56.8 ± 1.1	62.2 ± 1.1
Random	51.6 ± 1.1	49.8 ± 1.1	47.7 ± 1.2	46.3 ± 1.2

Table 6: EmbedLLM test Pass@1, with 95% bootstrap confidence intervals

Router / Pool size M	5	10	25	50	112
Oracle	77.2 ± 1.5	80.8 ± 1.4	89.6 ± 1.1	93.7 ± 0.9	98.0 ± 0.5
Capability Synthesis (ours)	63.5 ± 1.8	63.8 ± 1.8	**64.3 ± 1.8**	**64.3 ± 1.8**	**66.8 ± 1.6**
EmbedLLM MF	**63.6 ± 1.8**	63.7 ± 1.8	**64.3 ± 1.7**	64.0 ± 1.7	66.1 ± 1.7
Avengers	**63.6 ± 1.7**	**63.9 ± 1.7**	63.7 ± 1.8	63.7 ± 1.7	66.0 ± 1.6
MLP Classifier	60.7 ± 1.7	60.8 ± 1.7	60.7 ± 1.7	60.7 ± 1.7	62.1 ± 1.7
GraphRouter	60.3 ± 1.7	60.5 ± 1.7	60.6 ± 1.7	60.7 ± 1.7	60.8 ± 1.7
Best single model	60.7 ± 1.7	60.7 ± 1.7	60.7 ± 1.7	60.7 ± 1.7	60.7 ± 1.7
Weighted random	58.8 ± 1.5	58.5 ± 1.5	57.1 ± 1.4	56.3 ± 1.4	53.1 ± 1.3
Random	57.4 ± 1.5	56.0 ± 1.5	52.0 ± 1.3	48.7 ± 1.3	40.7 ± 1.0

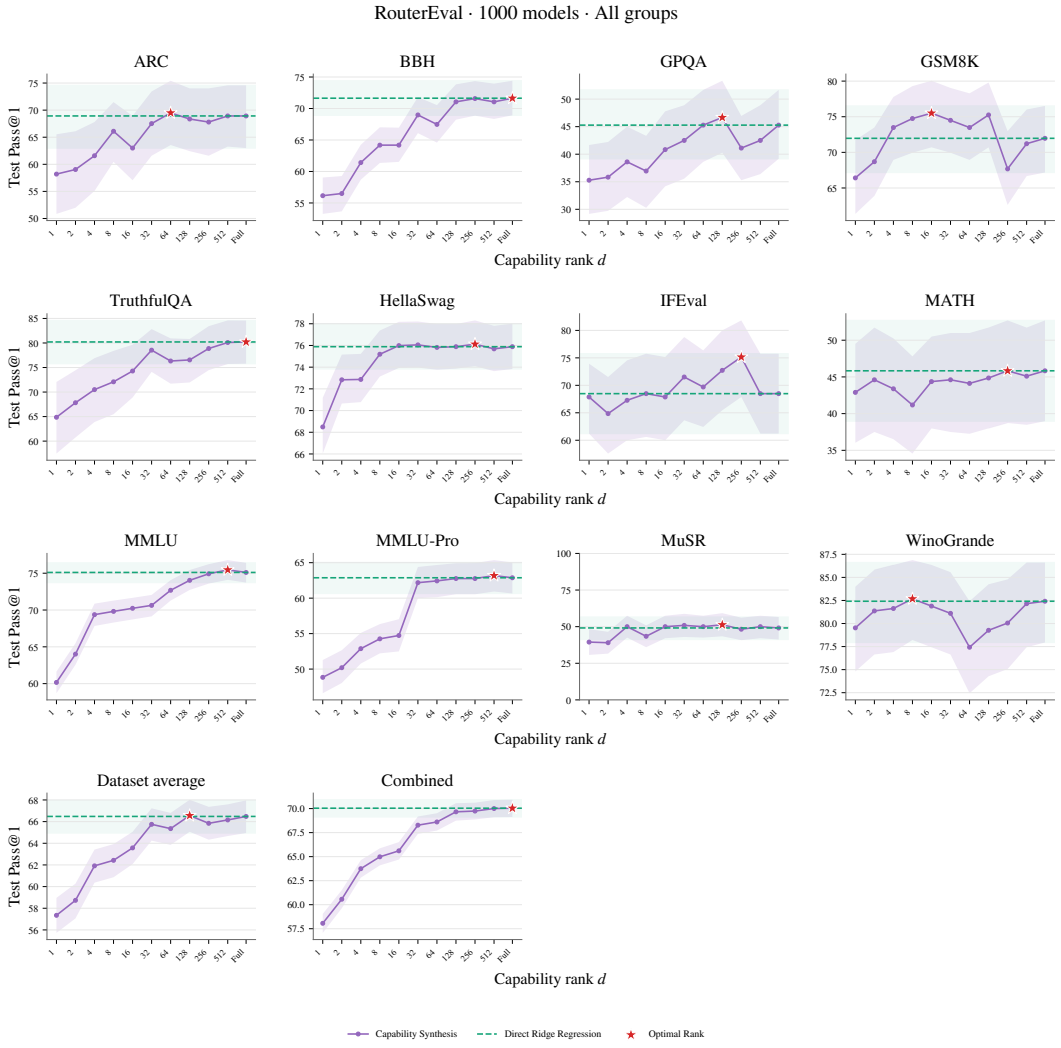


Figure 5: RouterEval Rank Ablation. Pool Size $M = 1000$

C.2 Rank Ablations

We provide rank ablations of the individual datasets of RouterEval and EmbedLLM as well as the combined dataset in Figures 5 and 6. In each plot, we report the test pass@1 of Capability Synthesis across different ranks.

C.3 Baselines Configurations

Table 7 compares the number of fitted router parameters for Capability Synthesis and EmbedLLM MF across different pool sizes and benchmarks. Here, M is the pool size and d the Capability Synthesis rank selected on the validation split. Capability Synthesis counts $\theta \in \mathbb{R}^{D\times d}$ and $P \in \mathbb{R}^{M\times d}$; EmbedLLM MF counts the text projection, the model embeddings (dimension 232), and the 2-way classifier. The frozen text encoder is excluded because it is common accross both of the methods. For RouterEval, we report the median for rank, and mean for number of parameters over 12 datasets and 3 pool groups.

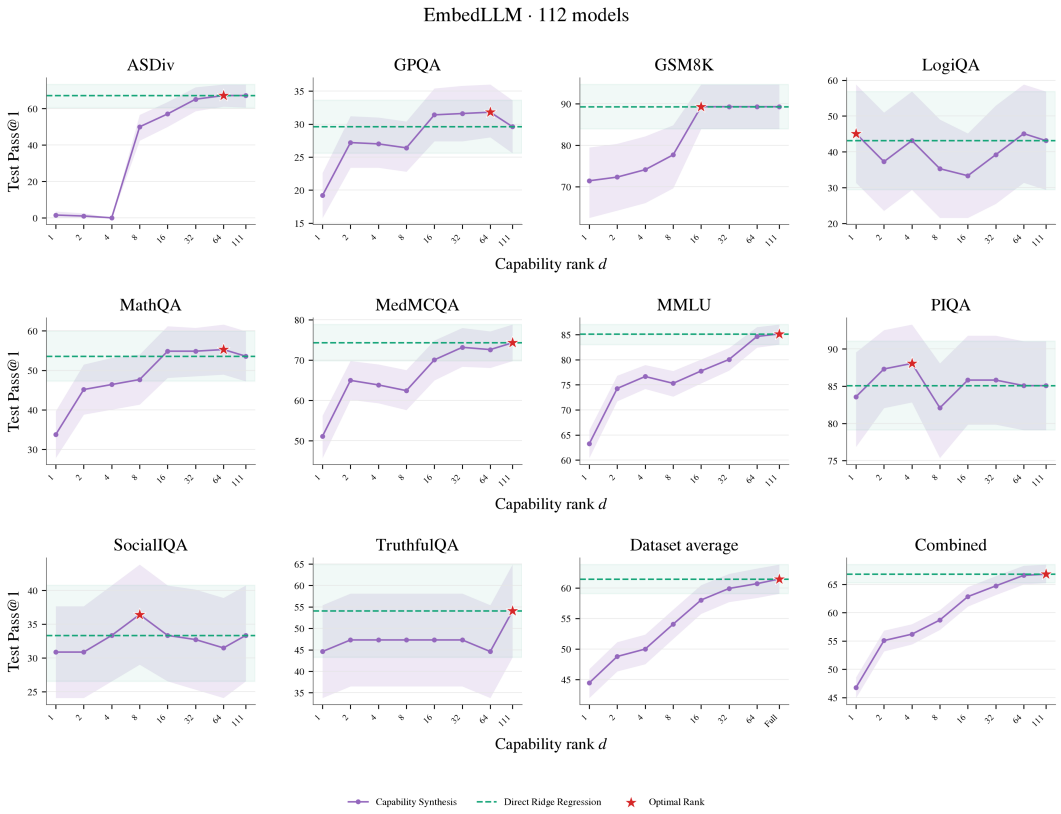


Figure 6: EmbedLLM Rank Ablation. Pool Size $M = 112$

Table 7: Number of fitted router parameters for Capability Synthesis and EmbedLLM MF. M is the pool size and d the Capability Synthesis rank selected on the validation split.

Benchmark	M	d	Capability Synthesis	EmbedLLM MF	Ratio (MF / CS)
EmbedLLM ($D = 768$)	5	4	3,092	180,034	58.2×
	10	9	7,002	181,194	25.9×
	25	24	19,032	184,674	9.7×
	50	49	40,082	190,474	4.8×
	112	111	97,680	204,858	2.1×
RouterEval ($D = 768$)	5	2	2,018	180,034	89.2×
	10	4	3,242	181,194	55.9×
	100	32	31,682	202,074	6.4×
	1000	112	324,133	410,874	1.3×

Table 8: 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	$d \in [\min(M-1, N)]^\star$
	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