ROUTING STARTS WITH REPRESENTATION: BEHAVIORAL QUERY PRETRAINING FOR LLM ROUTERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) vary substantially in capability, making model routing a promising approach to improving inference efficiency by selecting an appropriate model for each query. Existing work has largely focused on designing more capable routers, while treating the query representation as a fixed input from a general-purpose text encoder. We show that this assumption is limiting: semantic query geometry is systematically misaligned with the behavioral geometry relevant to routing. Motivated by this observation, we introduce behavioral query pretraining, which uses historical query–model interactions to fine-tune a query encoder with multi-model supervision and then discards the prediction head, yielding a reusable routing-aware representation. Across diverse frozen-query routing methods, the resulting representation consistently improves routing performance and is effective under limited target supervision.

1 INTRODUCTION

Large language models (LLMs) vary substantially in capability (Brown et al., 2020; Chowdhery et al., 2022; Touvron et al., 2023; Jiang et al., 2023), making model routing a promising approach to improving inference efficiency by selecting an appropriate model for each query (Hu et al., 2024; Ong et al., 2025; Feng et al., 2026). Many existing approaches build routing mechanisms on top of fixed query representations, typically obtained from a general-purpose text embedding model (Hu et al., 2024; Zhuang et al., 2025; Feng et al., 2025). We argue that this design warrants closer examination: before increasing router complexity, one should first ask whether the router is given an appropriate representation.

Semantic embeddings organize queries by meaning (Reimers & Gurevych, 2019; Gao et al., 2021), whereas routing requires a different notion of similarity: two queries are similar if they induce similar preferences over candidate models. We find that these two geometries are systematically misaligned. Queries that are close in behavioral space exhibit substantially more consistent model preferences than semantic neighbors. This suggests that generic semantic embeddings fail to expose model-discriminative information that is directly useful for routing.

Motivated by this observation, we introduce *behavioral query pretraining*, a simple approach for learning routing-aware query representations from historical query–model interactions. Given the outcomes of multiple models on each query, we fine-tune a pretrained query encoder with a multi-model behavioral prediction objective. The resulting encoder is frozen and reused across downstream routers. Routing-relevant structure is thus learned directly in the query representation rather than left entirely to the downstream routing architecture.

Our experiments show that behavioral query representations improve all frozen-query router families we evaluate. The gains become consistent across routers once modest target supervision is available and persist as the supervision budget increases. On RouterBench, behavioral representations also improve the quality–cost trade-off under recorded API costs. Despite substantial changes in local geometry, coarse task-level semantic information remains nearly perfectly recoverable.

Our contributions are threefold. First, we identify a mismatch between semantic query geometry and the model-preference structure required for routing. Second, we propose behavioral query pretraining, which converts historical query–model interactions into a reusable frozen query representa-

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.

tion. Third, we show consistent gains across routers and supervision budgets, improved quality–cost trade-offs, and preservation of coarse semantic structure.

2 Related Work

2.1 LLM Routing

LLM routing exploits capability and cost heterogeneity across language models by selecting an appropriate model for each query (Chen et al., 2024a; Ding et al., 2024; Šakota et al., 2024; Aggarwal et al., 2024; Lu et al., 2024). Existing methods include classifiers, pairwise preference predictors, matrix-factorization approaches, retrieval-based routers, and graph-based models, with recent work further considering cost-aware routing, dynamic model pools, and unseen-model generalization (Ong et al., 2025; Feng et al., 2025; Jitkrittum et al., 2025; Varangot-Reille et al., 2026). Despite these architectural differences, many frozen-query routers first encode the query with a general-purpose text embedding model and then learn the routing rule on top of this fixed representation (Hu et al., 2024; Ong et al., 2025; Zhuang et al., 2025).

Our work targets this upstream component rather than the routing architecture itself. We ask whether generic query embeddings expose the model-discriminative information required for routing, and learn a behavioral representation that can be reused by otherwise unchanged linear, neural, factorization-based, retrieval-based, and structured routers.

2.2 Routing-Aware Representation Learning

Recent routing methods increasingly learn structured representations of candidate models, using behavioral profiles, latent model embeddings, or joint query–model representations to capture capability differences (Zhuang et al., 2025; Chen et al., 2024b; Jitkrittum et al., 2025; Ong et al., 2025; Hu et al., 2024). In these approaches, representation learning (Bengio et al., 2014; Hadsell et al., 2006; Khosla et al., 2020) is primarily organized around the model side (Zhang et al., 2025b), while query representations are commonly inherited from generic text encoders or optimized only within a particular router (Reimers & Gurevych, 2019; Gao et al., 2021).

Our work instead learns a reusable query-side representation from multi-model behavioral supervision and discards the source-model prediction head after pretraining. This follows the broader principle that task-relevant supervision can reshape representations beyond generic similarity objectives (Weinberger & Saul, 2009; Khosla et al., 2020; Burges et al., 2005; Caruana, 1997). The resulting frozen query encoder can be reused across different model representations and downstream routing mechanisms.

3 Problem Setup and Motivation

3.1 LLM Routing through Query Representations

Consider a set of queries $\mathcal{Q}$ and a candidate pool of M language models $\mathcal{M} = \{m_1, \ldots, m_M\}$. For each query $q \in \mathcal{Q}$, let

$$\mathbf{y}_q = [y_{q1}, \ldots, y_{qM}] \in \mathbb{R}^M \tag{1}$$

denote the observed outcomes of the candidate models, where y_{qi} measures the performance of model m_i on query q. A router aims to select an appropriate model for each query,

$$r(q, \mathcal{M}) \to m_i, \tag{2}$$

possibly subject to additional considerations such as inference cost.

Most learned routers do not operate directly on the query text. Instead, a query encoder E first maps the query to a representation $\mathbf{z}_q = E(q)$, which is then consumed by a downstream routing function:

$$r(E(q), \mathcal{M}) \to m_i. \tag{3}$$

This decomposition exposes two distinct sources of routing capacity. The query representation determines what information about the query is made available to the router, whereas the downstream

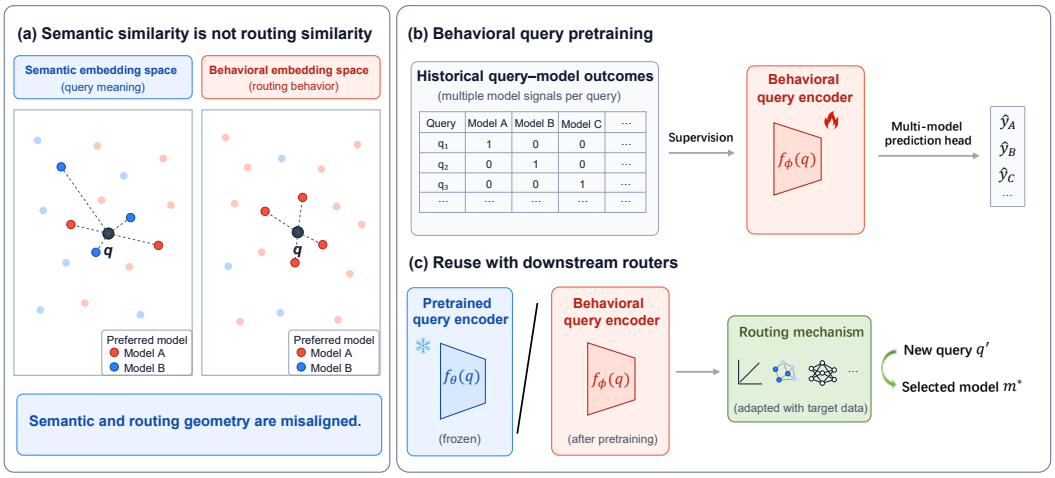

Figure 1: **Overview of behavioral query pretraining for LLM routing.** **(a)** Semantic similarity does not necessarily reflect routing similarity: queries that are close in a generic embedding space can induce different model preferences, whereas behavioral similarity organizes queries according to their observed model-performance patterns. **(b)** We use historical query–model outcomes as multi-model supervision to adapt a pretrained query encoder. **(c)** At deployment, the behavioral encoder is frozen and replaces the generic semantic encoder as a drop-in query representation for downstream routing mechanisms, which are trained only with target-side supervision.

routing mechanism determines how that information is converted into model preferences or selection decisions. Existing work has primarily increased the capacity of the latter, while the former is often inherited from a general-purpose text embedding model and treated as fixed.

This design implicitly assumes that a representation optimized for semantic similarity already exposes the query properties needed to distinguish candidate models. However, increasing router capacity cannot fully compensate for model-discriminative information that is poorly expressed in its input representation. This motivates a more fundamental question: *does generic semantic query geometry align with the geometry required for model routing?*

3.2 Semantic Similarity Is Not Routing Similarity

We begin by directly testing whether generic semantic geometry aligns with the geometry required for model routing. Semantic embeddings regard two queries as similar when their textual representations are close, whereas routing suggests a different criterion: queries should be considered similar when they induce similar relative preferences among candidate models.

We compare these two notions using disjoint subsets of models. For each random split, half of the models are designated as source models and the remaining half as target models. A query's behavioral neighborhood is constructed solely from its binary correctness profile over the source models, while its semantic neighborhood is constructed from the frozen Qwen3 query embedding. We then evaluate both neighborhoods on the target models by measuring *pair-preference agreement*: whether neighboring queries tend to prefer the same model within a target-model pair. The source and target model sets are disjoint, so this experiment tests whether query similarity inferred from one collection of models predicts routing preferences among another collection. Formal definitions are provided in Appendix A.

Table 1 shows a clear and consistent difference. Across three independent random 50/50 model splits, behavioral neighborhoods consistently exhibit substantially higher target-model pair-preference agreement than semantic neighborhoods. At the default neighborhood size $k = 10$, semantic neighborhoods achieve an average agreement of 0.618, whereas behavioral neighborhoods reach 0.783, an absolute improvement of 0.165. The same pattern holds across neighborhood sizes. At $k = 5$, pair-preference agreement increases from 0.628 to 0.797 ($+0.169$), while at $k = 20$ it

Table 1: **Semantic versus behavioral neighborhoods.** Pair-preference agreement is averaged over three random 50/50 source–target model splits. We report the mean and standard deviation across splits. The final column shows the behavioral improvement over the semantic neighborhood.

k	Semantic	Behavioral	Δ	Seed std of Δ
5	0.6279 ± 0.0082	0.7966 ± 0.0034	+0.1687	0.0116
10	0.6180 ± 0.0073	0.7832 ± 0.0027	+0.1651	0.0084
20	0.6101 ± 0.0072	0.7747 ± 0.0017	+0.1646	0.0088

increases from 0.610 to 0.775 ($+0.165$). Thus, the observed mismatch is not sensitive to a particular choice of neighborhood size.

These results indicate that semantic proximity and routing proximity capture substantially different query structure. Moreover, the behavioral structure inferred from one subset of models generalizes to relative preferences among unseen models. This suggests that historical query–model interactions contain model-discriminative information that is poorly exposed by generic semantic representations. We therefore ask whether such information can be absorbed into the query encoder itself, motivating the behavioral query pretraining method introduced next.

4 Behavioral Query Pretraining

The mismatch identified in Section 3.2 suggests that historical query–model interactions contain routing-relevant structure that is only weakly exposed by generic semantic representations. We therefore propose *behavioral query pretraining*, which uses multi-model outcomes as supervision to adapt the query encoder itself. The central design principle is to separate *learning a reusable query representation* from *learning a particular routing policy*. Rather than tying representation learning to a specific router or candidate model pool, we first train the encoder to predict historical model behavior and then reuse the resulting representation with otherwise unchanged downstream routers.

Multi-model behavioral supervision. Let E_{θ_0} denote a pretrained text encoder, with

$$\mathbf{z}_q^{\mathrm{raw}} = E_{\theta_0}(q) \in \mathbb{R}^d \tag{4}$$

the original representation of query q. Given a source query set $\mathcal{Q}_{\mathrm{src}}$ and source model pool

$$\mathcal{M}_{\mathrm{src}} = \{m_1, \ldots, m_M\}, \tag{5}$$

we observe a behavioral matrix

$$\mathbf{Y} = [y_{qi}] \in \{0,1\}^{|\mathcal{Q}_{\mathrm{src}}| \times M}, \tag{6}$$

where y_{qi} indicates whether model m_i answers query q correctly. The corresponding row

$$\mathbf{y}_q = [y_{q1}, \ldots, y_{qM}] \tag{7}$$

forms a multi-model behavioral signature for the query. Unlike a scalar difficulty score or a single best-model label, $\mathbf{y}_q$ preserves which models succeed and fail jointly, thereby retaining information about relative capability structure across the model pool.

We train the query encoder through a lightweight multi-model prediction head. For

$$\mathbf{z}_q = E_\theta(q), \tag{8}$$

we first project the representation through a low-dimensional bottleneck,

$$\mathbf{h}_q = \mathrm{GELU}\left(\mathbf{W}_b \mathbf{z}_q + \mathbf{b}_b\right), \tag{9}$$

where GELU is used as the activation function (Hendrycks & Gimpel, 2023). The model then predicts one logit for each source model,

$$\mathbf{s}_q = \mathbf{W}_p \mathbf{h}_q + \mathbf{b}_p \in \mathbb{R}^M, \tag{10}$$

where s_{qi} predicts the observed outcome y_{qi}. We jointly optimize the encoder and prediction layers with binary cross-entropy,

$$\mathcal{L}_{\text{beh}} = -\frac{1}{|\Omega|} \sum_{(q,i)\in\Omega} \left[y_{qi} \log \sigma(s_{qi}) + (1 - y_{qi}) \log(1 - \sigma(s_{qi})) \right], \tag{11}$$

where Ω denotes the set of observed query–model interactions. This masked formulation naturally supports incomplete interaction matrices.

Crucially, all model outcomes must be predicted from the same query representation $\mathbf{z}_q$. The encoder is therefore trained to expose query properties that jointly predict how different models behave, rather than properties optimized for a single model or routing decision.

Learning a reusable behavioral representation. The bottleneck and prediction layers serve only as a supervisory interface during pretraining; they are not part of the downstream router. We parameter-efficiently adapt E_{θ_0} using LoRA (Hu et al., 2021), obtaining $E_{\theta_{\text{beh}}}$. After pretraining, both $\mathbf{W}_b$ and $\mathbf{W}_p$ are discarded. The only transferred component is the behavioral query encoder,

$$\mathbf{z}_q^{\text{beh}} = E_{\theta_{\text{beh}}}(q). \tag{12}$$

Downstream routing therefore requires neither source-model identities nor predictions from the pretraining head. This distinction separates our method from approaches that directly learn a routing policy over $\mathcal{M}_{\text{src}}$: the transferred object is a query representation rather than a policy tied to the source candidate set. Without behavioral pretraining, each downstream router must infer routing-relevant structure from a generic semantic representation using only target-side supervision. Our approach instead learns this structure at the representation level from large-scale historical query–model interactions. The same encoder can subsequently be reused across different routing architectures and candidate-model pools.

Reuse representation with downstream routers. Consider a target candidate pool $\mathcal{M}_{\text{tgt}}$ and a downstream router with parameters ϕ. We write its model score generically as

$$s_\phi(\mathbf{z}_q, m), \qquad m \in \mathcal{M}_{\text{tgt}}, \tag{13}$$

with routing decision

$$\hat{m}(q) = \arg\max_{m\in\mathcal{M}_{\text{tgt}}} s_\phi(\mathbf{z}_q, m). \tag{14}$$

Behavioral pretraining changes only the query-side input,

$$\mathbf{z}_q^{\text{raw}} \longrightarrow \mathbf{z}_q^{\text{beh}}, \tag{15}$$

while leaving the routing function and its target-side training procedure unchanged. The behavioral encoder remains frozen, and only ϕ is learned from target-side supervision.

The query encoder applies to a broad range of routing mechanisms considered in the LLM-routing literature (Hu et al., 2024; Ong et al., 2025; Feng et al., 2025; Jitkrittum et al., 2025; Varangot-Reille et al., 2026). For example, a linear router scores each candidate model as

$$s_\phi^{\text{lin}}(\mathbf{z}_q, m) = \mathbf{w}_m^\top \mathbf{z}_q + b_m, \tag{16}$$

whereas an MLP can jointly predict scores for the entire candidate set,

$$\mathbf{s}_\phi^{\text{mlp}}(q) = \mathbf{W}_2 \rho(\mathbf{W}_1 \mathbf{z}_q + \mathbf{b}_1) + \mathbf{b}_2. \tag{17}$$

Factorization-based routers instead introduce a learned representation $\mathbf{e}_m$ for each model and score query–model compatibility through

$$s_\phi^{\text{mf}}(\mathbf{z}_q, m) = (\mathbf{W}_q \mathbf{z}_q)^\top \mathbf{e}_m, \tag{18}$$

as in matrix-factorization formulations for model routing (Ong et al., 2025; Hu et al., 2024). Here the query representation is consumed through an explicit interaction with a learned model embedding.

The same interface also covers non-parametric and profile-based routers. A KNN router estimates model performance from neighboring labeled queries,

$$s^{\mathrm{knn}}(\mathbf{z}_q, m) = \frac{1}{k} \sum_{q' \in \mathcal{N}_k(\mathbf{z}_q)} y_{q'm}, \tag{19}$$

as in retrieval-based routing (Hu et al., 2024; Varangot-Reille et al., 2026). Because no learned transformation is applied before neighbor retrieval, performance under KNN directly probes the local geometry induced by the query representation.

For profile-based routing, let $\boldsymbol{\mu}_c \in \mathbb{R}^{|\mathcal{M}_{\mathrm{tgt}}|}$ denote the model-performance profile associated with cluster c, and $a_\phi(c \mid \mathbf{z}_q)$ the assignment of query q to that cluster. The resulting score is

$$s_\phi^{\mathrm{profile}}(\mathbf{z}_q, m) = \sum_c a_\phi(c \mid \mathbf{z}_q)\mu_{c,m}, \tag{20}$$

where a_ϕ may be either a hard cluster assignment or a learned query-to-cluster mapping, corresponding to the UniRoute-style variants used in our experiments (Jitkrittum et al., 2025).

Finally, graph-based routers jointly model query–model interactions rather than scoring each pair independently (Feng et al., 2025; 2026). They can be written as structured scorers in which $\mathbf{z}_q$ initializes the query node of a query–model interaction graph,

$$s_\phi^{\mathrm{graph}}(\mathbf{z}_q, m) = D_\phi\left(H_\phi\left(G; \mathbf{z}_q, \mathbf{e}_m\right)\right)_{q,m}, \tag{21}$$

where H_ϕ denotes graph message passing and D_ϕ denotes an edge decoder.

Implementation. We use Qwen3-Embedding-0.6B (Zhang et al., 2025a) as the base encoder, with $d = 1024$. The behavioral prediction head consists of a 64-dimensional GELU bottleneck followed by an M-way linear prediction layer. We adapt the attention projection matrices of the encoder using LoRA (Hu et al., 2021). The behavioral prediction head is discarded after pretraining, and all downstream experiments use the resulting encoder in frozen form unless stated otherwise.

5 Experiments

5.1 Experimental Setup

EmbedLLM. Our main experiments use EmbedLLM, a dense query–model interaction benchmark containing 29,917 queries and 112 language models. Each query–model pair is associated with a binary correctness outcome. For each of three seeds, we independently partition the models into 89 seen and 23 held-out models, and partition queries into approximately 70%/10%/20% training, validation, and test sets. Behavioral pretraining observes only training-query outcomes on seen models; test-query and held-model outcomes are never exposed during representation training.

Representations. Raw denotes the frozen Qwen3-Embedding-0.6B representation using last-token pooling and ℓ_2 normalization. Behavior uses the same encoder after behavioral query pretraining (Section 4).

Routers. We evaluate eight frozen-query routing variants: logistic regression (LogReg), a two-layer MLP, pairwise and pointwise matrix factorization (MF-Pair and MF-Point), two UniRoute variants based on K-means and a learned query-to-cluster map, and k-nearest neighbors (KNN). These cover linear and nonlinear prediction heads, learned model representations, pairwise supervision, behavioral model profiles, and non-parametric retrieval. For every Raw–Behavior comparison, all router-side components and training protocols are held fixed.

Metrics. Our primary EmbedLLM metric is routing reward. Given router scores $s(q, m)$, we select from the model pool $\mathcal{M}$ by

$$\hat{m}(q) = \arg\max_{m \in \mathcal{M}} s(q, m). \tag{22}$$

Table 2: **Behavioral representations across frozen-query routers.** Mean routing reward and macro AUC over three seeds; ΔR and ΔAUC denote Behavior minus Raw.

	Routing reward			Macro AUC		
Downstream Router	Raw	Behavior	ΔR	Raw	Behavior	ΔAUC
KNN	0.5982	0.6014	+0.0031	0.6981	0.7251	+0.0270
LogReg	0.6466	0.6515	+0.0049	0.7183	**0.7441**	+0.0257
MLP	0.6371	0.6487	+0.0115	0.7238	0.7438	+0.0200
GraphRouter	0.5788	0.5926	+0.0138	0.6223	0.7298	**+0.1075**
MF-Pair	0.6406	0.6504	+0.0098	0.5839	0.6014	+0.0175
MF-Point	0.6368	0.6466	+0.0099	0.7141	0.7434	+0.0293
UniRoute-KMeans	0.6281	**0.6564**	**+0.0283**	0.6941	0.7344	+0.0403
UniRoute-LearnedMap	0.6422	0.6456	+0.0033	0.7247	0.7384	+0.0138

and compute routing reward as

$$R = \frac{1}{|\mathcal{Q}_{\text{test}}|} \sum_{q \in \mathcal{Q}_{\text{test}}} y_{q,\hat{m}(q)}. \tag{23}$$

We additionally report *macro AUC*: we treat the outcome of each model as a separate binary prediction task: for each model m, we compute the AUC of $s(q, m)$ for predicting y_{qm} across test queries, and then average uniformly over models,

$$\text{AUC}_{\text{macro}} = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} \text{AUC}\left(\{s(q,m)\}_q, \{y_{qm}\}_q\right). \tag{24}$$

Thus, macro AUC measures how well the router identifies the queries on which each individual model is likely to succeed, without allowing frequently successful models to dominate the metric.

External cost-aware evaluation. We additionally evaluate frozen transfer to RouterBench, which contains a disjoint pool of 11 API models and records the realized API cost of each query–model call. Crucially, the Behavior encoder used in this experiment is pretrained only on EmbedLLM and is never updated using RouterBench outcomes. We train only lightweight downstream routers on the target benchmark. Besides routing quality, we report normalized quality–cost area (NQCA) and matched-quality API-cost savings.

5.2 Behavioral Representations Benefit Diverse Frozen Routers

We first isolate representation quality by freezing the query encoder and changing only its output representation. Table 2 compares Raw and Behavior under eight downstream routers spanning linear, nonlinear, factorization-based, retrieval-based, cluster-based, and graph-based routing mechanisms. For every comparison, the router architecture, training data, and optimization protocol are kept fixed, so the only change is the query representation.

Behavior improves both mean routing reward and macro AUC for all eight routers, showing that the gain is not tied to a particular downstream architecture or inductive bias. The largest routing-reward improvement is obtained by UniRoute-KMeans ($\Delta R = +0.0283$), while GraphRouter exhibits the largest macro-AUC gain ($\Delta\text{AUC} = +0.1075$). These consistent improvements suggest that behavioral pretraining exposes routing-relevant information upstream, reducing the extent to which individual routers must recover such structure from generic semantic embeddings.

5.3 Behavioral Representations Improve Few-Shot Routing

We next evaluate whether behavioral pretraining remains useful when only a limited number of labeled target queries are available. We train the downstream router using only K labeled queries ($K \in \{8, 16, 32, 64, 128, 256, 512, 1024\}$) while keeping the query encoder frozen. Raw and Behavior use the same router architecture and training protocol, so the only difference is the query representation supplied to the router.

Table 3: **Few-shot routing with frozen query representations.** Routing reward under different numbers of labeled target queries K, averaged over three seeds. Each entry reports RAW / BEHAVIOR; the better result is shown in bold.

K	LogReg	KNN	MLP	MF-Pair	UniRoute-KMeans
8	0.4903 / **0.5019**	0.4928 / 0.4928	**0.4831** / 0.4777	0.4454 / **0.4707**	0.4657 / **0.4702**
16	0.5071 / **0.5170**	0.5047 / 0.5047	**0.5136** / 0.4942	0.4324 / **0.4733**	0.4546 / **0.4888**
32	0.5438 / **0.5511**	0.5317 / **0.5437**	**0.5216** / 0.5138	0.4676 / **0.5087**	0.5169 / **0.5178**
64	0.5383 / **0.5622**	0.5406 / **0.5576**	**0.5273** / 0.5225	0.5373 / **0.5590**	0.5266 / **0.5422**
128	0.5295 / **0.5664**	0.5473 / **0.5701**	0.5426 / **0.5434**	0.5651 / **0.5713**	0.5470 / **0.5758**
256	0.5279 / **0.5745**	0.5627 / **0.5915**	0.5384 / **0.5604**	0.5637 / **0.5833**	0.5878 / **0.6184**
512	0.5301 / **0.5816**	0.5762 / **0.6066**	0.5451 / **0.6058**	0.5691 / **0.6073**	0.6048 / **0.6395**
1024	0.5404 / **0.5859**	0.5831 / **0.6112**	0.5639 / **0.6175**	0.5716 / **0.6108**	0.6118 / **0.6426**

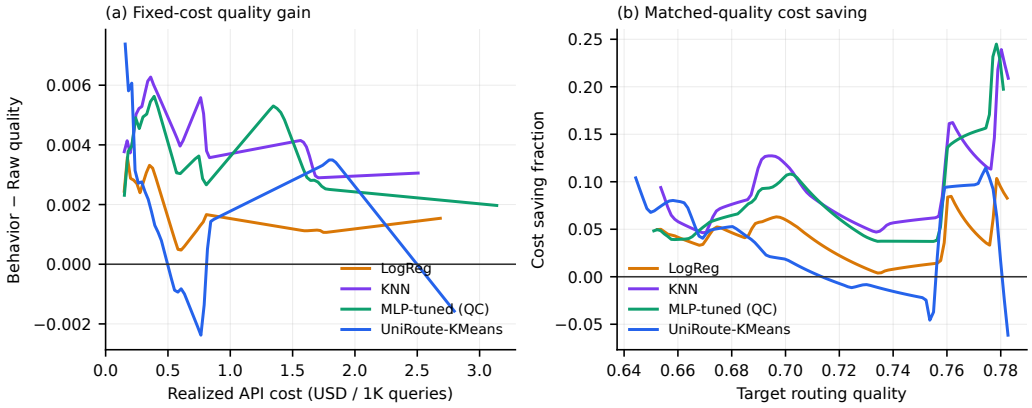

Figure 2: **Behavioral representations improve quality–cost trade-offs on RouterBench.** (a) Difference in routing quality between Behavior and Raw at matched realized API cost. Positive values indicate higher quality at the same cost. (b) Fraction of API cost saved by Behavior at matched routing quality. Curves are computed from the Pareto upper envelope and averaged over three seeds.

Table 3 reveals a clear supervision-dependent pattern. At very small budgets, downstream estimation noise can obscure the benefit of the representation, particularly for MLP. Once $K \geq 128$, however, BEHAVIOR outperforms RAW in all 20 router–budget combinations. The average gain across routers also grows with supervision, from $+0.019$ at $K = 128$ to $+0.043$ at $K = 512$, and remains $+0.039$ at $K = 1024$. This suggests that behavioral pretraining provides routing-relevant structure that becomes increasingly exploitable as the downstream router receives enough target supervision.

5.4 BEHAVIORAL REPRESENTATIONS IMPROVE THE QUALITY–COST FRONTIER

Routing quality alone does not capture the practical objective of model routing: achieving strong predictions without unnecessarily invoking expensive models. We therefore evaluate the transferred representations on RouterBench, where each query–model call has a recorded API cost. The behavioral encoder is pretrained only on EmbedLLM and remains frozen on RouterBench; only the downstream router is trained on the target benchmark.

We compare Raw and Behavior over their complete shared quality–cost range. For each API-cost budget, the router selects the highest-scoring affordable model. We characterize the resulting frontier using three complementary metrics: fixed-cost quality gain, matched-quality cost saving, and normalized quality–cost area (NQCA), where higher NQCA indicates better average quality at a given cost.

Figure 2 shows that the gains persist across the operating range rather than only at the unconstrained point. Behavior dominates the full shared cost range for KNN and MLP. The effect is less uniform

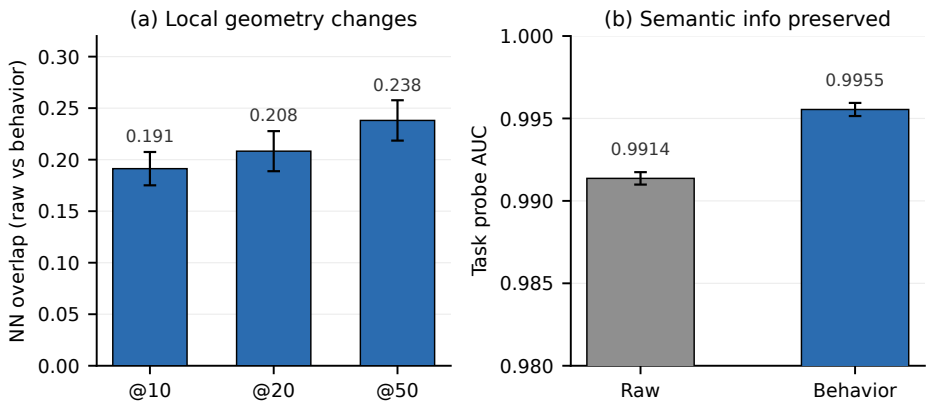

Figure 3: **Behavioral pretraining reorganizes local geometry while preserving task-level semantics. (a)** Raw and Behavior share few nearest neighbors. **(b)** Task-category information remains nearly perfectly linearly recoverable.

for LogReg and UniRoute-KMeans, whose frontiers cross locally, but Behavior is still better over roughly 72their shared cost ranges.

The integrated metrics show the same pattern. As shown in Table 4, behavior improves NQCA for all evaluated routers, with gains ranging from +0.0010 to +0.0038 (Table 4). These results show that a behavioral encoder trained only on EmbedLLM transfers to a disjoint RouterBench model pool and improves not only routing quality but also quality–cost efficiency, rather than merely shifting decisions toward more expensive models.

5.5 Behavioral Pretraining Reorganizes Rather Than Removes Semantics

Finally, we test whether improved routing structure comes at the expense of the original semantic information. As shown in Figure 3 Raw and Behavior share only 0.1913, 0.2082, and 0.2381 of their nearest neighbors at $k = 10$, 20, and 50, respectively, showing substantial local reorganization. At the same time, a linear probe for the original EmbedLLM task categories remains nearly perfect: using the dataset-provided `category` labels, AUC changes only from 0.9914 under Raw to 0.9955 under Behavior. Although local neighborhoods change substantially, information about the underlying task identity remains linearly accessible, while model-relative capability requirements become more accessible to downstream routers.

Table 4: Cost-aware routing performance on RouterBench. We report normalized quality–cost area (NQCA), averaged over three seeds. Δ denotes the improvement from replacing the raw query representation with our behavioral representation.

Router	Raw	Behavior	Δ
LogReg	0.7602	0.7617	+0.0015
KNN	0.7579	0.7617	+0.0038
MLP	0.7599	0.7624	+0.0026
MF-Pair	0.7601	0.7611	+0.0010
UniRoute-KMeans	0.7554	0.7569	+0.0015
UniRoute-LearnedMap	0.7560	0.7582	+0.0023
Average	0.7582	0.7607	+0.0025

6 Conclusion

We show that query representation is an important component of LLM routing. Behavioral query pretraining converts historical query–model interactions into a reusable representation that consistently improves diverse frozen-query routers across supervision budgets. It also improves quality–cost trade-offs while preserving coarse semantic information. These results suggest that improving the representation supplied to the router can be as important as increasing router complexity.

7 AI USE STATEMENT

In this work, we used generative AI tools for the following required-disclosure tasks: proposing or refining hypotheses; assisting with translation; designing or providing feedback on research methodology or experiments; supporting qualitative and thematic data analysis; and interpreting results. We have not used generative AI tools for any other required-disclosure tasks, including generating synthetic data sets; helping develop theoretical models or conceptual frameworks; formulating mathematical claims; providing critical ingredients for proving mathematical claims; assisting in the writing of proofs; implementing methods; and cleaning and reformatting datasets. No remaining required-disclosure tasks are applicable to this work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. Automix: Automatically mixing language models. In *NeurIPS*, 2024.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives, 2014.

Tom B. Brown, Benjamin Mann, Nick Ryder, et al. Language models are few-shot learners. In *NeurIPS*, 2020.

Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. Learning to rank using gradient descent. In *ICML*, 2005.

Rich Caruana. Multitask learning. *Machine Learning*, 1997.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024a.

Shuhao Chen, Weisen Jiang, Baijiong Lin, James T. Kwok, and Yu Zhang. RouterDC: Query-based router by dual contrastive learning for assembling large language models. In *NeurIPS*, 2024b.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *ICLR*, 2024.

Tao Feng, Yanzhen Shen, and Jiaxuan You. GraphRouter: A graph-based router for LLM selections. In *ICLR*, 2025.

Tao Feng, Fangxu Yu, Haozhen Zhang, Zhongjie Dai, Liangqi Yuan, Zijie Lei, Weizhi Zhang, Kunlun Zhu, Haodong Yue, Keyang Xuan, Ge Liu, and Jiaxuan You. Llmrouter: Unified infrastructure for developing, evaluating, and deploying llm routers, 2026.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In *EMNLP*, 2021.

R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

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

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In *ACL*, 2023.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Zifeng Wang, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon, and Sanjiv Kumar. Universal model routing for efficient LLM inference. *arXiv preprint arXiv:2502.08773*, 2025.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *NeurIPS*, 2020.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. Routing to the expert: Efficient reward-guided ensemble of large language models. In *NAACL*, 2024.

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

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *EMNLP-IJCNLP*, 2019.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Clovis Varangot-Reille, Christophe Bouvard, and Antoine Gourru. Generalising LLM routing using past performance retrieval: A few-shot router is sufficient. In *EACL SRW*, 2026.

Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 2009.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models, 2025a.

Yi-Kai Zhang, De-Chuan Zhan, and Han-Jia Ye. Capability instruction tuning: A new paradigm for dynamic llm routing. In *AAAI*, 2025b.

Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. EmbedLLM: Learning compact representations of large language models. In *ICLR*, 2025.

Marija Šakota, Maxime Peyrard, and Robert West. Fly-swat or cannon? cost-effective language model choice via meta-modeling. In *WSDM*, 2024.

A SEMANTIC–BEHAVIORAL NEIGHBORHOOD COMPARISON

We provide the full definition of the neighborhood experiment used in Section 3.2.

Data and model splits. Let

$$Y \in \{0,1\}^{N\times M} \tag{25}$$

denote the query–model correctness matrix, where $N = 29{,}917$ queries and $M = 112$ models. For each query q and model m, $Y_{qm} = 1$ indicates that model m answers query q correctly.

We repeat the experiment using three random seeds $\{42, 2024, 7\}$. For each seed, the 112 models are randomly permuted and divided evenly into a source set $\mathcal{M}_S$ and target set $\mathcal{M}_T$:

$$|\mathcal{M}_S| = |\mathcal{M}_T| = 56, \qquad \mathcal{M}_S \cap \mathcal{M}_T = \varnothing. \tag{26}$$

The source models are used only to construct behavioral neighborhoods, whereas the target models are used only to evaluate neighborhood agreement.

Semantic neighborhoods. Each query is represented by the frozen 1024-dimensional Qwen3-Embedding-0.6B embedding,

$$\mathbf{z}_q^{\text{sem}} \in \mathbb{R}^{1024}. \tag{27}$$

We ℓ_2-normalize these embeddings and compute cosine similarity between queries. The semantic neighborhood of query q is

$$\mathcal{N}_k^{\text{sem}}(q) = \text{TopK}_{q'\neq q}\ \cos\left(\mathbf{z}_q^{\text{sem}}, \mathbf{z}_{q'}^{\text{sem}}\right). \tag{28}$$

Because this representation does not depend on the model split, the semantic neighbors are shared across the three seeds.

Behavioral neighborhoods. For each model split, query q is represented by its binary correctness profile over the 56 source models:

$$\mathbf{z}_q^{\text{beh}} = [Y_{qm}]_{m\in\mathcal{M}_S} \in \{0,1\}^{56}. \tag{29}$$

We again apply ℓ_2 normalization and cosine similarity and define

$$\mathcal{N}_k^{\text{beh}}(q) = \text{TopK}_{q'\neq q}\ \cos\left(\mathbf{z}_q^{\text{beh}}, \mathbf{z}_{q'}^{\text{beh}}\right). \tag{30}$$

All nearest-neighbor similarities are computed in FP32. Since $\mathcal{M}_S$ changes with the model-split seed, behavioral neighborhoods are recomputed independently for each split.

Target-model preference. For a pair of target models (m_i, m_j), we define the query-specific directional preference as

$$\pi_q^{(i,j)} = \text{sign}\left(Y_{qi} - Y_{qj}\right). \tag{31}$$

Hence,

$$\pi_q^{(i,j)} = \begin{cases} +1, & Y_{qi} = 1,\ Y_{qj} = 0, \\ -1, & Y_{qi} = 0,\ Y_{qj} = 1, \\ 0, & Y_{qi} = Y_{qj}. \end{cases} \tag{32}$$

A tie, $\pi_q^{(i,j)} = 0$, contains no directional information about the relative preference between the two models and is therefore excluded from the corresponding comparison.

Pair-preference agreement. For an anchor query q, target-model pair (m_i, m_j), and neighborhood $\mathcal{N}_k(q)$, let

$$\widetilde{\mathcal{N}}_k^{(i,j)}(q) = \left\{q' \in \mathcal{N}_k(q) : \pi_{q'}^{(i,j)} \neq 0\right\} \tag{33}$$

be the subset of neighbors that have a non-tied preference for the same target pair.

If the anchor itself has a non-tied preference and $|\widetilde{\mathcal{N}}_k^{(i,j)}(q)| > 0$, we define

$$A_k(q; i, j) = \frac{\sum_{q'\in\widetilde{\mathcal{N}}_k^{(i,j)}(q)} \mathbf{1}\left[\pi_{q'}^{(i,j)} = \pi_q^{(i,j)}\right]}{|\widetilde{\mathcal{N}}_k^{(i,j)}(q)|}. \tag{34}$$

Thus, $A_k(q; i, j)$ is the fraction of valid neighbors that prefer the same target model as the anchor query.

Each target set contains

$$\binom{56}{2} = 1540 \tag{35}$$

model pairs. For a given model split and neighborhood type, the reported agreement is the macro-average of Eq. 34 over all valid anchor-query–target-pair units:

$$A_k = \mathrm{mean}_{(q,i,j)\in\mathcal{V}}\, A_k(q; i, j), \tag{36}$$

where $\mathcal{V}$ contains all units for which the anchor preference is non-tied and at least one neighbor has a non-tied preference.

We denote the resulting statistics by

$$A_k^{\mathrm{sem}} \qquad \text{and} \qquad A_k^{\mathrm{beh}}, \tag{37}$$

and report their difference

$$\Delta A_k = A_k^{\mathrm{beh}} - A_k^{\mathrm{sem}}. \tag{38}$$

Aggregation over model splits. We compute the complete statistic independently for each of the three 50/50 model splits and report the arithmetic mean and sample standard deviation across seeds.

At the primary setting $k = 10$, the three per-split improvements are all positive:

$$0.1662,\ 0.1729,\ 0.1563. \tag{39}$$

Their average is

$$\Delta A_{10} = 0.1651. \tag{40}$$

A paired query-level bootstrap with 5,000 replicates gives a 95% confidence interval of

$$[0.1633,\ 0.1670]. \tag{41}$$

B Behavioral Pretraining Details

We provide the implementation details for the behavioral query pretraining described in Section 4. Unless otherwise stated, the same configuration is used across all three random seeds.

Encoder and input processing. We initialize the query encoder from Qwen3-Embedding-0.6B (Zhang et al., 2025a), whose hidden dimension is $d = 1024$. Queries are truncated to a maximum sequence length of 256 tokens. Following the original embedding model, we use the representation of the last token as the query embedding and apply ℓ_2 normalization to the extracted embedding before downstream routing.

Parameter-efficient adaptation. We adapt the encoder using LoRA (Hu et al., 2021). LoRA modules are inserted into the query, key, value, and output projections of every attention layer, i.e., `q_proj`, `k_proj`, `v_proj`, and `o_proj`. We use rank $r = 16$, scaling factor $\alpha = 32$, and LoRA dropout 0.1. This introduces approximately 3.15M trainable LoRA parameters while leaving the original encoder weights frozen.

Behavioral prediction head. For behavioral pretraining, the 1,024-dimensional encoder output is passed through a 64-dimensional bottleneck with GELU activation, followed by a linear prediction layer with one output per source model:

$$1024 \to 64 \to |\mathcal{M}_{\mathrm{src}}|. \tag{42}$$

On EmbedLLM, $|\mathcal{M}_{\mathrm{src}}| = 89$ for each model split. The bottleneck and prediction head contain approximately 71K trainable parameters and are used only during behavioral pretraining. Both components are discarded after training; downstream experiments use only the adapted query encoder.

Table 5: Hyperparameters for behavioral query pretraining.

Hyperparameter	Value
Base encoder	Qwen3-Embedding-0.6B
Encoder dimension	1024
Maximum sequence length	256
Pooling	Last token
Output normalization	ℓ_2 normalization
LoRA rank r	16
LoRA scaling α	32
LoRA dropout	0.1
LoRA target modules	`q/k/v/o_proj`
Trainable LoRA parameters	≈ 3.15M
Bottleneck dimension	64
Bottleneck activation	GELU
Prediction-head outputs	89
Bottleneck + head parameters	$\approx$ 71K
Optimizer	AdamW
Encoder learning rate	2×10^{-5}
Bottleneck/head learning rate	10^{-3}
Weight decay	0.01
Batch size	32
Maximum epochs	5
LR schedule	Warmup + cosine decay
Objective	Multi-label BCE

Optimization. We optimize the LoRA parameters and behavioral prediction head jointly with AdamW. The encoder-side LoRA parameters use a learning rate of 2×10^{-5}, while the bottleneck and prediction head use a learning rate of 10^{-3}. We set weight decay to 0.01, batch size to 32, and train for at most 5 epochs. The learning rate follows a warmup followed by cosine decay schedule. Training minimizes the multi-label binary cross-entropy objective in Eq. 11.

Training protocol and data isolation. For each seed, behavioral pretraining uses only the query-training split and the 89 source models assigned to that seed. The corresponding validation-query split is used for checkpoint selection. Neither test-query outcomes nor outcomes from the 23 held-out models are accessed during behavioral pretraining. We use seeds 42, 2024, and 7 for the three independent model/query splits. After checkpoint selection, the prediction head is removed and embeddings for downstream experiments are extracted using the frozen behavioral encoder. Thus, all downstream Raw–Behavior comparisons differ only in the query representation supplied to the router.