Scalable Prompt Routing via Fine-Grained Latent Task Discovery

Yunyi Zhang, Soji Adeshina, Patrick Guan, Ashwin Ganesh, Zhen Han,
Vassilis N. Ioannidis, Huzefa Rangwala, George Karypis
Amazon Web Services
zhyunyi@amazon.com

Abstract

Prompt routing dynamically selects the most appropriate large language model from a pool of candidates for each query, optimizing performance while managing costs. As model pools scale to include dozens of frontier models with narrow performance gaps, existing approaches face significant challenges: manually defined task taxonomies cannot capture fine-grained capability distinctions, while monolithic routers struggle to differentiate subtle differences across diverse tasks. We propose a two-stage routing architecture that addresses these limitations through automated fine-grained task discovery and task-aware quality estimation. Our first stage employs graph-based clustering to discover latent task types and trains a classifier to assign prompts to discovered tasks. The second stage uses a mixture-of-experts architecture with task-specific prediction heads for specialized quality estimates. At inference, we aggregate predictions from both stages to balance task-level stability with prompt-specific adaptability. Evaluated on 10 benchmarks with 11 frontier models, our method consistently outperforms existing baselines and surpasses the strongest individual model while incurring less than half its cost.

1 Introduction

Large language models (LLMs) exhibit diverse capabilities across different task types, with no single model consistently outperforming all others. This heterogeneity motivates *prompt routing*, aiming to dynamically select the most appropriate model from a candidate pool for each query to optimize performance while managing computational costs. As model pools expand to include dozens of powerful candidates, a fundamental challenge emerges: how can routing methods accurately distinguish fine-grained capability differences across models and task types at scale?

Existing prompt routing approaches face significant limitations when scaling to large model pools with narrow performance gaps. Most methods either rely on manually defined coarse-grained task taxonomies (NVIDIA, 2024) or train monolithic routers that predict model quality across all prompt types (Ong et al., 2025; Feng et al., 2025a; Chen et al., 2024a; Ding et al., 2024). The former approach becomes infeasible as manual taxonomy design cannot keep pace with the nuanced strengths of LLMs, while the latter struggles to capture fine-grained distinctions when a single estimator must differentiate subtle capability differences across diverse tasks. For instance, within the broad category of "mathematics," models may exhibit vastly different performance on symbolic algebraic manipulation versus contextual word problems, yet coarse categorization treats these uniformly. Furthermore, when routing among frontier models with narrow performance gaps, the routing task becomes substantially more challenging, requiring the system to identify subtle task-model affinity patterns that determine which model is *best* for a given prompt.

We propose a two-stage routing architecture, FineRouter, that leverages automated fine-grained task discovery and task-aware quality estimation. Rather than forcing a monolithic model to handle all distinctions simultaneously, we explicitly infer latent task structure, allowing specialized components to focus on specific task types. Our Stage 1 develops an offline graph-based clustering method that automatically discovers fine-grained task types from training data. For each discovered task type, we adaptively select top candidate models and train a classifier to efficiently assign task types to input prompts during inference. Stage 2 employs a mixture-of-experts quality estimation architecture where task-specific prediction heads are invoked based on the assigned task type. This design enables specialized routing knowledge for each task type while maintaining computational efficiency. At inference, our router combines complementary signals from both stages, balancing task-level sta-

bility with instance-level adaptability.

We evaluate our approach on 10 diverse benchmarks spanning various tasks, routing among 11 state-of-the-art frontier models including Claude-Sonnet-4.5, DeepSeek-R1, Llama-4-Maverick, and Qwen3-235B. Our method consistently outperforms existing routing baselines and achieves superior performance compared to any individual model, including surpassing the strongest candidate while incurring less than half its inference cost. Ablation studies confirm that both stages contribute meaningfully to overall performance, with fine-grained task discovery providing more effective routing signals than coarse-grained taxonomies. Case studies reveal that our clustering method successfully identifies meaningful task distinctions that align with known model capabilities while also discovering unexpected niche domains.

Our main contributions are as follows:

- A scalable automated task discovery method that combines semantic and performance-based signals to identify fine-grained task types from large-scale training data.
- A task-aware routing architecture employing mixture-of-experts quality estimation with specialized prediction heads that leverage discovered task structure for more accurate model selection.
- Comprehensive evaluation on 10 benchmarks with 11 frontier models as candidates, demonstrating consistent improvements over baselines and superior cost-performance tradeoffs compared to single-model deployment.

2 Preliminaries

Prompt Routing Let $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ denote a set of n candidate large language models. Given an input prompt p, the goal of prompt routing is to select the most appropriate model $M^* \in \mathcal{M}$ that optimizes a desired objective.

Quality-Based Routing Most existing prompt routing methods frame this as a single-class classification task which predicts one LLM that performs the best on the input prompt (Ong et al., 2025; Feng et al., 2025b). However, we argue that as the LLMs become more powerful, there are increasingly more cases where multiple models perform similarly well. These subtle distinctions in their performance cannot be captured by a coarse classification output and thus makes the classification setting brittle. Therefore, we adopt Quality-Based Routing (Feng et al., 2025a) which turns routing into a regression task. Formally, assume there is a quality function $Q^* : (p, r) \to \mathbb{R}$ that assigns a quality score to each pair of prompt and LLM-generated response, $r = M(p)$. Empirically Q^* can be any task-specific evaluation functions or general reward models (Liu et al., 2025). Then the best-performing model selection can be defined as:

$$M^*(p) = \arg\max_{M_i \in \mathcal{M}} Q^*(p, M_i(p)). \quad (1)$$

However, evaluating all models at inference time is computationally prohibitive and counterproductive to the routing formulation. Therefore, the routing problem requires learning a quality estimator $\tilde{Q}$ that predicts the best model without generating responses from all candidates,

$$\mathcal{R}(p) = \arg\max_{M_i \in \mathcal{M}} \tilde{Q}(p, M_i). \quad (2)$$

Task-Aware Routing Objective As n grows large, learning an accurate quality estimator $\tilde{Q}$ becomes challenging. Different models excel at different task types, and a monolithic estimator struggles to capture these fine-grained distinctions. This motivates our approach of discovering latent task structure to enable task-aware quality estimation. To address this challenge, we introduce the concept of latent task types. Let $\mathcal{T} = \{t_1, t_2, \ldots\}$ represent a set of discovered task types. Our goal is to learn both a task assignment function $t : p \to \mathcal{T} \cup \{\emptyset\}$ and a task-aware routing function,

$$\mathcal{R}_t(p) = \arg\max_{M_i \in \mathcal{M}} \tilde{Q}_t(p, M_i). \quad (3)$$

that leverages task-specific knowledge to improve routing accuracy while maintaining computational efficiency at inference time.

3 Methodology

Figure 1 shows an overview of our two-stage routing architecture. Stage 1 identifies fine-grained task types through graph-based clustering and trains a classifier to assign new prompts to discovered tasks. Stage 2 employs an MoE quality estimation model with task-specific prediction heads. At inference, predictions from both stages are aggregated to produce the final model selection.

3.1 Stage 1: Task Type Discovery and Matching

The first stage of our routing architecture automatically discovers fine-grained task types from training data and learns to match new prompts to these discovered tasks. Unlike existing works that rely on pre-defined coarse-grained task taxonomies (NVIDIA, 2024; Feng et al., 2025b), our

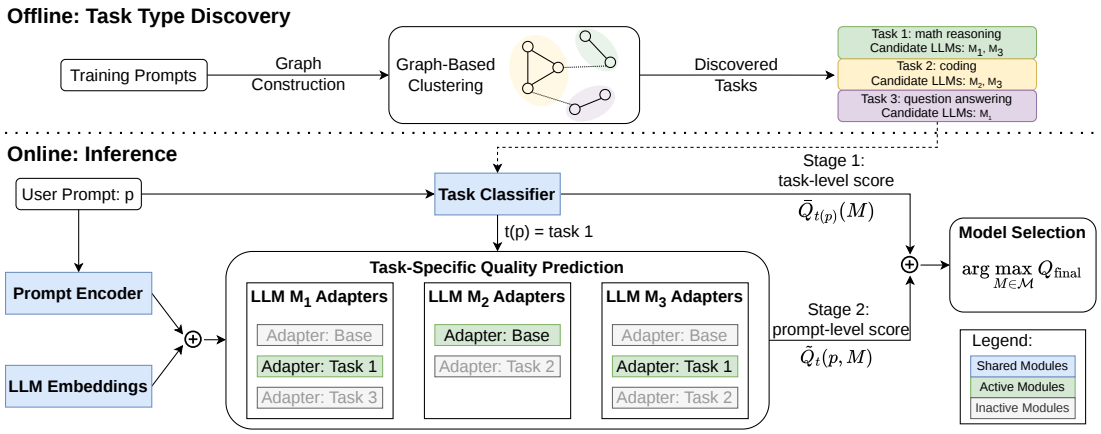

Figure 1: Overview of FineRouter. **Top**: Offline task type discovery via graph-based clustering, producing fine-grained tasks with candidate LLMs per task. **Bottom**: Online inference where the task classifier assigns prompts to discovered tasks, enabling task-specific adapter activation in the MoE router. Final model selection aggregates task-level scores (Stage 1) with prompt-specific quality predictions (Stage 2).

method automates the discovery of fine-grained task structure directly from data. This automation is crucial for two reasons: (1) manual task taxonomy design becomes infeasible as model pools scale to hundreds of candidates with nuanced strengths, and (2) data-driven discovery can reveal latent task distinctions that may not be apparent through manual categorization but are critical for effective model differentiation.

3.1.1 Task Type Discovery

Given a training set covering diverse prompts from different sources, we first use an LLM to generate a concise sentence describing the task for each prompt.[1] These task descriptions provide semantic representations that capture the nature of each prompt. We then apply a graph-based clustering method that combines two complementary signals: (1) semantic similarity between task descriptions, and (2) similarity in model preference patterns as reflected by ranked lists of preferred LLMs.

For a prompt p, let $\pi_p = [M_1, M_2, \ldots, M_n]$ denote a ranked list of LLMs from the model pool $\mathcal{M}$, where models are ordered by decreasing preference on their responses according to the quality function, $Q^*(p, M_i(p))$. The rank of model M in list π_p is denoted as $\text{rank}_{\pi_p}(M)$.

Graph Construction We construct a sparse prompt graph where nodes represent individual prompts and edges encode both semantic and performance-based similarity. First, we identify k-nearest neighbors for each prompt based on cosine similarity between task description embeddings. For each candidate edge, we compute a pairwise Rank Biased Overlap (RBO) (Webber et al., 2010) score between the ranked lists of preferred LLMs for the two prompts. We filter out edges with RBO scores below a threshold τ, retaining only pairs that exhibit similar model preferences. For the remaining edges, we set edge weights to the geometric mean of normalized cosine similarity and RBO scores after min-max normalization.

Iterative Clustering We apply Leiden community detection (Traag et al., 2019) to identify prompt clusters. For each detected community $C = \{p_1, p_2, \ldots, p_{|C_j|}\}$, we (1) compute a cluster center through median pooling of its task description embeddings and (2) construct a combined ranked list of preferred LLMs using rank fusion across all prompts in the cluster using the mean reciprocal rank (MRR) score,

$$\text{MRR}_C(M) = \frac{1}{|C|} \sum_{p_i \in C} \frac{1}{\text{rank}_{\pi_p}(M) + \epsilon}, \tag{4}$$

where ϵ is a constant (typically $\epsilon = 60$) that reduces the impact of high-rank differences. This summarization enables recursive application of the community detection algorithm.

This iterative process continues for ℓ iterations to refine the clusters. We only keep the clusters from the final iterations, where each cluster $\mathcal{C}_i$ represents a discovered task type t_i consisting of semantically similar prompts that share similar preferred LLMs. Importantly, our method only clusters prompts that form meaningful task communities. Prompts that do not fit into any cluster are treated as not belonging to a specific task type.

Candidate Model Selection For each discovered

[1] We show our prompt in Appendix A.

task cluster $\mathcal{C}_i$, we identify a small set of top candidate LLMs $\mathcal{L}_i$ that are most likely to perform well on that specific task. We use the same rank fusion process to combine the ranked lists of preferred LLMs across all prompts within the cluster. Instead of using a fixed top-k parameter, we adaptively choose the number of candidate LLMs to maximize the coverage of the candidate set over the cluster's preferred models. Specifically, let $\text{Cov}(\mathcal{L}_i)$ be the frequency of prompts within the cluster whose most preferred LLM appears in $\mathcal{L}_i$. Then we incrementally increase the number of candidate models until $\text{Cov}(\mathcal{L}_i)$ exceeds a predefined threshold δ. This adaptive selection ensures that each task type is associated with a focused set of strong candidate models.

3.1.2 Task Type Classifier

To match prompts with discovered task types at inference time, we train a text classifier that predicts matching scores between prompts and task types, denoted as $t : p \rightarrow \mathcal{T} \cup \{\emptyset\}$. The classifier employs a bi-linear matching architecture consisting of two components: (1) a prompt encoder initialized from a pre-trained text encoder, and (2) task type encodings initialized with embeddings of LLM-generated summarizing task descriptions for each prompt cluster. This architecture enables efficient computation of matching scores between prompt and task type representations.

Given the large number of target classes (typically hundreds of discovered tasks), we fine-tune the classifier in a multi-label setting using binary cross-entropy loss. This formulation allows prompts to potentially match multiple task types with varying confidence scores, providing flexibility in task assignment during inference.

3.2 Stage 2: Task-Aware Dynamic Router

With the task type classifier from Stage 1, we can now assign any prompt to one of the discovered fine-grained task types (or to no specific task if all matching scores are low). Building on these task assignments, Stage 2 employs a task-aware routing mechanism. Instead of training a single monolithic router across all prompt types, we leverage the discovered task structure to enable specialized routing decisions through a mixture-of-experts architecture. Specifically, we utilize a mixture-of-experts quality estimation architecture where specialized prediction heads are invoked based on the predicted task type of the incoming prompt.

3.2.1 Model Architecture

Our router model consists of three main components: (1) a prompt encoder initialized from a pre-trained transformer-based encoder, (2) an LLM embedding layer that maps model IDs to model-specific representations, and (3) a Quality Estimation (QE) layer with task-aware prediction heads. Input prompts and LLMs are encoded with (1) and (2) respectively, which are then concatenated and passed to the QE layer to predict an estimated quality score $\tilde{Q}_t(p, M)$.

The QE layer implements a mixture-of-experts architecture with two types of MLP-based prediction heads. First, we maintain $|\mathcal{M}|$ general adapters (MLPs), with each adapter predicting the quality score for one specific model. These general adapters learn to estimate each model's expected response quality based on global knowledge acquired from the entire training data, providing baseline predictions applicable across all prompt types.

Second, for each discovered task type t_i with the selected candidate models $\mathcal{L}_i$ identified in Stage 1, we initialize $|\mathcal{L}_i|$ task-specific quality prediction adapters. Because these candidate LLMs are most likely to perform well on the corresponding task, the task-specific adapters can learn specialized knowledge that better predicts their performance on this particular task type. All adapter heads share the same prompt encoder and LLM embeddings, ensuring efficient parameter usage while enabling specialized predictions.

If a prompt is assigned to task type t_i, we invoke the task-specific adapters for models in $\mathcal{L}_i$ and the general adapters for all other models in $\mathcal{M} - \mathcal{L}_i$. This hybrid invocation strategy combines the benefits of task-specific expertise with comprehensive coverage: the task-specific adapters provide refined predictions for the most promising candidates based on learned task patterns, while the general adapters ensure that potentially strong models outside the selected candidates are still considered, preventing the router from prematurely excluding viable options.

3.2.2 Model Training

We train the router model to mirror its inference-time behavior. First, we re-label the entire training set using the task type classifier obtained from Stage 1. During training, task-specific prediction heads are trained only on prompts assigned to their corresponding task type, allowing each expert to specialize in its designated task do-

main. We optimize all prediction heads using mean squared error (MSE) loss between the predicted quality scores $\tilde{Q}_t(p, M)$ and the ground-truth quality scores $Q^*(p, M(p))$.

For more effective training, we adopt a two-phase approach. We first train the base model consisting of the prompt encoder, LLM embedding layer, and general quality prediction heads on all training data. Then, we fine-tune the task-specific prediction heads while freezing the prompt encoder and LLM embeddings on the task-type-labeled training data. This staged training strategy ensures that the shared representations remain stable while task-specific experts learn to refine predictions for their specialized domains.

3.3 Inference

At inference time, we deploy both stages to produce the final routing decision through a two-step process that combines task-based prior knowledge with prompt-specific quality estimation.

First, we apply the task type classifier to assign a task type $t_i \in \mathcal{T} \cup \{\emptyset\}$ to the incoming prompt p. If the prompt is assigned to a specific task t_i, we invoke the corresponding task-specific adapters for models in $\mathcal{L}_i$ along with general adapters for models in $\mathcal{M} - \mathcal{L}_i$ to obtain quality estimates $\tilde{Q}_t(p, M)$ for all models. If no task assignment is made (i.e., $t(p) = \emptyset$), we use only the general adapters to predict quality scores across all models.

To leverage complementary information from both stages, we aggregate their predictions through a weighted combination. Stage 1 provides task-based prior knowledge through the aggregated quality scores of each model on the assigned prompt cluster, denoted as $\bar{Q}_{t_i}(M)$, which represents the median quality score of model M across all prompts in cluster $\mathcal{C}_i$. Note that for prompts where $t(p) = \emptyset$, we construct an 'Others' cluster from all such training prompts and compute their aggregated median scores $\bar{Q}_\emptyset(M)$ as stage-1 scores. Stage 2 provides fine-grained, prompt-specific quality estimates $\tilde{Q}_t(p, M)$. We normalize both sets of scores to the range $[0, 1]$ using min-max normalization (denoted as $\text{norm}(\cdot)$) and compute the final routing score as:

$$Q_{\text{final}}(p, M) = \alpha \cdot \text{norm}(\tilde{Q}_t(p, M)) + (1 - \alpha) \cdot \text{norm}(\bar{Q}_{t(p)}(M)), \quad (5)$$

where $\alpha \in [0, 1]$ controls the relative weight between prompt-specific and task-based predictions. The final model selection is then:

$$R_t(p) = \arg\max_{M \in \mathcal{M}} Q_{\text{final}}(p, M). \quad (6)$$

This aggregation strategy enables the router to benefit from both the stability of task-level patterns and the adaptability of prompt-specific predictions, resulting in more robust routing decisions.

Importantly, this inference process maintains computational efficiency: the task type classifier requires only a single forward pass, and the adaptive activation of prediction heads ensures that the effective model size during inference remains constant regardless of the number of discovered tasks.

4 Experiments

4.1 Experimental Setup

Datasets We evaluate our approach on 10 benchmark datasets that cover a wide range of natural language understanding and reasoning tasks: question answering (NQ (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), CommensenseQA (Talmor et al., 2019)), multiple-choice (MMLU (Hendrycks et al., 2021b,a), ARC-Challenge (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018)), mathematical reasoning (GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021c)), and code generation (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021)). We split the combined dataset into 278,977 training samples, 34,872 development samples, and 34,873 test samples.

Model Candidates We evaluate our routing approach across a diverse set of 11 recent state-of-the-art language models spanning multiple model families and capability profiles. Our candidate pool includes models from the Llama family (LLAMA-3.3-70B (Llama Team, 2024), LLAMA-4-MAVERICK[2]), Anthropic's Claude series (CLAUDE-HAIKU-4.5[3], CLAUDE-SONNET-4.5[4]), Mistral AI models[5] (MISTRAL-LARGE, MISTRAL-SMALL), DeepSeek (DEEPSEEK-V3 (DeepSeek-AI, 2025b), DEEPSEEK-R1 (DeepSeek-AI, 2025a)), Qwen3 models (Qwen-Team, 2025) (QWEN3-32B, QWEN3-235B-A22B-THINKING), and OpenAI's open-source model GPT-OSS-120B (OpenAI, 2025).

[2] https://ai.meta.com/blog/llama-4-multimodal-intelligence/

[3] https://www.anthropic.com/claude/haiku

[4] https://www.anthropic.com/claude/sonnet

[5] https://docs.mistral.ai/getting-started/models

Table 1: Performance comparison across 10 benchmarks. Best model and best router scores are **boldfaced**.

Models	Per-Task Evaluation										Overall	
	NQ	Triv-QA	Com-QA	mmlu	arc	OB-QA	gsm8k	math	hum-eval	mbpp	Ave	Qual
Candidate Models												
Llama-3.3-70B	57.6	23.2	76.0	87.4	90.5	90.7	89.2	94.0	21.1	40.0	67.0	0.446
Llama-4-Maverick	53.8	22.7	79.9	93.0	**98.8**	95.7	**94.0**	**95.2**	21.1	86.7	74.1	0.580
Claude-Haiku-4.5	46.6	21.6	79.8	85.2	**98.8**	95.5	83.8	88.1	**73.7**	86.7	76.0	0.572
Claude-Sonnet-4.5	59.4	24.7	**85.7**	**94.7**	97.6	**97.4**	88.7	91.1	63.2	**93.3**	**79.6**	**0.621**
Mistral-Small	50.3	22.3	59.2	74.1	69.1	67.2	74.1	25.1	31.6	26.7	50.0	0.409
Mistral-Large	56.0	24.8	75.8	70.5	92.9	85.1	84.4	71.0	57.9	23.3	64.2	0.489
DeepSeek-v3	**60.7**	25.7	70.7	89.2	91.7	84.0	91.9	89.0	36.8	73.3	71.3	0.412
DeepSeek-R1	60.3	**26.0**	81.0	92.0	**98.8**	94.4	90.4	91.6	47.4	86.7	76.9	0.551
Qwen3-32B	39.5	18.7	78.9	85.3	96.4	90.3	86.6	75.5	21.1	33.3	62.6	0.528
Qwen3-235B-A22B	55.1	23.7	85.2	93.9	91.7	90.9	90.6	93.3	52.6	83.3	76.0	0.530
GPT-OSS-120B	48.3	21.6	78.7	89.6	97.6	93.5	79.1	88.4	26.3	80.0	70.3	0.362
Routers												
kNN	56.2	24.4	79.8	91.9	97.6	94.6	90.3	92.2	42.1	66.7	73.6	0.620
MLP	53.6	23.8	60.2	74.7	70.2	68.2	75.4	59.5	63.2	80.0	62.9	0.449
RouteLLM	55.0	22.1	77.1	89.2	97.6	92.9	82.6	87.5	26.3	43.3	67.4	0.387
RouterDC	50.3	22.4	76.4	89.0	91.7	90.5	82.9	88.0	26.3	46.7	66.4	0.353
GraphRouter	48.3	21.7	78.7	90.1	**98.8**	93.5	79.7	89.6	26.3	80.0	70.7	0.386
IPR	57.6	24.3	81.7	92.2	97.7	95.3	90.4	93.6	47.4	83.3	76.3	0.646
FineRouter	**59.0**	**24.9**	**83.3**	**93.4**	**98.8**	**95.7**	**91.6**	**93.8**	**68.4**	**90.0**	**79.9**	**0.652**

Unlike previous works that primarily focus on smaller open-source models, we intentionally select frontier models to mimic real-world deployment scenarios where users seek to optimally leverage all available models. Routing among such high-performing models presents a significantly more challenging task, as the performance gaps between models are narrower and more nuanced.

Baselines We compare against several routing methods: (1) kNN: selects models based on embedding similarity to training examples, (2) MLP: classifies query embedding to the most proper candidate LLM, (3) RouteLLM (Ong et al., 2025): matrix factorization based routing that learns latent factors for prompts and models, (4) RouterDC (Chen et al., 2024b): dual contrastive learning based routing, (5) GraphRouter (Feng et al., 2025b): graph-based routing using prompt relationships to tasks and LLMs, (6) IPR (Feng et al., 2025a): a quality estimation-based routing approach by fine-tuning prompt and LLM encoding with per-LLM quality prediction adapters.

For all compared methods, we use a state-of-the-art reward model SKYWORK-REWARD-V2-LLAMA-3.1-8B (Liu et al., 2025) as the reference quality function $Q^*(p, M(p))$ after normalizing to range $[0, 1]$. Therefore, for classification-based methods, the ground truth label is just $M^*(p)$ for each training prompt (Eq. 1). While existing works typically use the downstream evaluation metric as quality function (e.g., Exact Match for QA tasks), most of such metrics are binary and induce lots of ties in scores for stronger candidate models. Selecting the "best performing model" for each training prompt will be biased to "first-appearing candidate," leading to all classification-based routers overfit to select one single model for all test samples. For complete implementation details, see Appendix B.

Evaluation Metrics We report the task-specific evaluation metric for each dataset following (Feng et al., 2025c) with their macro average, and a quality score using $Q^*(\cdot)$.

4.2 Experiment Results

Table 1 presents the performance of FineRouter compared to baselines and individual candidate models across 10 diverse benchmarks. First, no single LLM consistently achieves the best performance across all tasks, which strongly motivates the value of prompt routing. Second, baseline routing methods struggle to outperform strong individual models, likely because our candidate pool consists of state-of-the-art frontier models with narrow performance gaps, making it substantially harder to distinguish subtle capability differences compared to prior work focusing on smaller models with larger performance disparities. Third, FineRouter achieves the strongest overall performance, consistently outperforming baselines across the majority of tasks. This demonstrates that explicit fine-grained task discovery and task-aware routing successfully leverage the complementary strengths of different frontier models.

Table 2: Ablation study comparing stage-1 only, stage-2 only, and full two-stage architecture.

Models	Per-Task Evaluation										Overall	
	NQ	Triv-QA	Com-QA	mmlu	arc	OB-QA	gsm8k	math	hum-eval	mbpp	Ave	Qual
Stage 1 Only												
Embedding	58.5	24.8	**84.5**	92.5	95.2	95.5	88.7	**94.3**	42.1	86.7	76.3	0.609
Coarse CLS	**60.3**	**25.9**	80.4	92.5	**98.8**	95.0	**92.8**	93.0	63.2	**93.3**	79.5	0.623
Fine CLS	59.3	25.1	82.7	93.3	**98.8**	**95.7**	91.8	93.6	63.2	**93.3**	79.7	0.637
Stage 2 Only	57.4	24.3	81.8	92.2	97.6	95.5	90.7	94.0	52.6	83.3	77.0	0.647
FineRouter	59.0	24.9	83.3	**93.4**	**98.8**	**95.7**	91.6	93.8	**68.4**	90.0	**79.9**	**0.652**

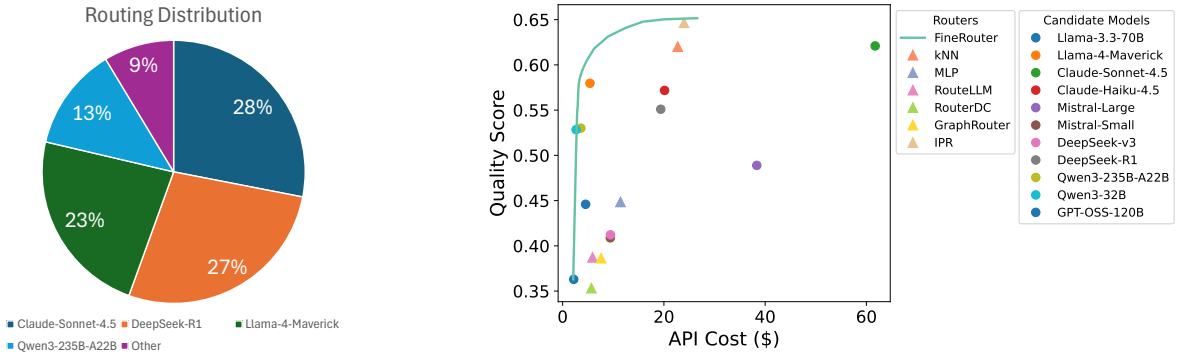

Figure 2: (a) Routing distribution of FineRouter across 11 candidate models. (b) Cost-performance comparison. FineRouter (curve) outperforms baseline routers (triangles) and individual LLMs (circles).

4.3 Routing Behavior Analysis

Figure 2(a) illustrates the routing distribution of FineRouter across the 11 candidate models on the test set. The router demonstrates balanced utilization across multiple high-performing models, with Claude-Sonnet-4.5 receiving 28%, DeepSeek-R1 27%, Llama-4-Maverick 23%, and Qwen3-235B 13% of prompts. This reflects the router's ability to identify and leverage each model's strengths across different task types, rather than over-relying on a single model.

Figure 2(b) presents the cost-performance trade-off comparing FineRouter, baseline routers, and individual LLMs. We plot the average quality score based on ground truth Q^* against the estimated total API cost[6] on all test samples. To obtain the cost-performance curve for routers, we follow (Feng et al., 2025a) to vary a tolerance parameter from 0 to 1, which controls the minimum performance the selected model should achieve comparing to the best score, all based on $Q_{\text{final}}(p, M)$ (Eq. 5). Our two-stage router consistently dominates baseline routers across all cost points while achieving superior performance compared to individual LLMs. Notably, FineRouter achieves better performance than the strongest candidate Claude-Sonnet-4.5 at less than half its cost, validating that task-aware routing enables better resource allocation. We also include a task classifier analysis in Appendix C.

[6]As of Feb 2026 through AWS Bedrock API: https://aws.amazon.com/bedrock/pricing/

4.4 Ablation Studies

To understand the contribution of each component, we conduct ablation studies comparing several variants. We include 3 variants for stage-1 only ablations: **Embeddings** uses prompt embeddings to match test prompts to the cluster centers and routes it to the top candidate model identified for the corresponding task. **Coarse CLS** uses a pretrained classifier with 10 pre-defined coarse-grained task types (NVIDIA, 2024), combined with our rank fusion method on our training data to determine recommended models for each task type. **Fine CLS** refers to our stage-1 task type classifier and routes each prompt to the top candidate LLM for the predicted task. **Stage-2 Only** uses the predicted scores from the stage-2 dynamic router without merging it with stage-1 scores.

Table 2 presents the ablation results. First, comparing Coarse CLS with Fine CLS demonstrates that our automatically discovered fine-grained task types provide more effective routing signals than manually pre-defined coarse-grained categories. Second, Stage-1-only approaches achieve competitive performance by leveraging discovered task structure to filter candidate models, validating the effectiveness of our task discovery method. Third, the full two-stage architecture consistently outperforms both Stage-1-only and Stage-2-only variants, demonstrating that task-aware specialized prediction heads provide complementary fine-grained routing signals beyond task-based candidate filter-

Table 3: Representative discovered task types showing fine-grained distinctions and model-specific strengths.

Class ID	Task Description	Selected Candidates
109	Answer multiple-choice questions requiring inference, detail identification, numerical calculation, and statement verification.	DeepSeek-v3
267	Formal symbolic mathematics: Solve algebra, geometry, number theory, linear algebra, and calculus problems.	Llama-4-Maverick, Llama-3.3-70B
161	Telephone area code queries: geographic locations, regional identifiers, and telecom history.	Qwen3-235B, GPT-OSS-120B, DeepSeek-R1

ing alone. These results confirm that both stages contribute meaningfully to the overall routing performance, with explicit task modeling enabling more accurate model selection decisions.

4.5 Case Studies: Task Discovery

Our Stage 1 clustering discovers 332 fine-grained task types with an average of 3.55 candidates per task, reducing the model pool to $\sim$32%. Table 3 presents representative examples. **Task 109** represents reading comprehension requiring numerical reasoning, with DeepSeek-v3 as the sole recommended model, aligned with its known strength in multi-step reasoning. **Task 267** reveals fine-grained distinctions within math: our method picks Llama-4-Maverick and Llama-3.3-70B for symbolic mathematics, while general word problems benefit from Claude-Sonnet-4.5 and DeepSeek-v3. This distinction would be missed by coarse-grained categorization that simply treats both as "math." **Task 161** exemplifies unexpected niche domains our method discovers: telephone area code queries combining geographic and telecom knowledge. Such fine-grained discoveries enable our router to make more informed decisions rather than treating all factual questions uniformly.

5 Related Work

Prompt routing aims to dynamically select the most appropriate model from a candidate pool for each query, balancing performance and cost. Early work by Chen et al. (2024a) introduced FrugalGPT, which cascades multiple LLMs and uses a learned scoring function to determine when to stop the cascade. HybridLLM (Ding et al., 2024) employs a BERT-based encoder to optimize cost-quality trade-offs by routing "easy" queries to smaller models and "hard" queries to larger models. RouteLLM (Ong et al., 2025) trains router models using human preference data and matrix factorization or BERT-based classifier to learn latent factors for prompts and models. Zooter (Lu et al., 2024) uses reward model scores as supervision signals and trains routers with RankNet loss, incorporating tag-based label enhancement to reduce reward model noise. RouterDC (Chen et al., 2024b) employs dual contrastive learning to train query-based routers that capture fine-grained distinctions between LLM capabilities. GraphRouter (Feng et al., 2025b) constructs a heterogeneous graph connecting queries to pre-defined coarse-grained task categories and LLMs, using edge prediction for routing. However, it relies on manually specified task taxonomies that cannot scale to capture the nuanced capability differences among frontier models. Feng et al. (2025a) developed IPR, a quality-constrained routing framework with modular architecture using per-LLM quality prediction adapters and user-controlled tolerance parameters for explicit cost-quality trade-offs. Our work differs from all prior approaches in two key ways: (1) we automatically discover task structure rather than relying on predefined categories, and (2) we use discovered tasks to specialize prediction heads rather than just filtering candidates.

6 Conclusion

We present a two-stage routing architecture, FineRouter, that addresses the challenge of scaling prompt routing to large pools of frontier models with narrow performance gaps. Our key contribution is the automated discovery of fine-grained latent task structure through graph-based clustering and a mixture-of-experts router that provides specialized quality estimates for each task type. Evaluated on 10 diverse benchmarks with 11 frontier models, our method consistently outperforms existing routing baselines and surpasses the strongest individual model while incurring less than half its inference cost. Our work demonstrates that explicit modeling of latent task structure enables more accurate and scalable routing decisions, particularly as model pools expand to include dozens of powerful candidates with subtle capability differences.

Limitations

While our approach demonstrates strong performance, several limitations merit discussion. First, our task discovery relies on LLM-generated task descriptions and reward model scores as supervision signals. The quality of discovered task types depends on the capability of the LLM used for description generation (we use Claude-Sonnet-4.5) and the accuracy of the reward model (Skywork-Reward-V2). These dependencies may introduce biases or miss task distinctions that are not well-captured by current reward models, particularly for specialized domains or creative tasks where reward modeling remains challenging.

Second, our graph-based clustering requires several hyperparameters (k-nearest neighbors, RBO threshold τ, coverage threshold δ, number of iterations ℓ) that were tuned on our training data. While we provide default values that work well across our benchmarks, optimal settings may vary for different domains or model pools. The clustering process also assumes that prompts with similar semantic descriptions and model preferences form meaningful task communities, which may not hold for all prompt distributions.

Third, our two-stage architecture requires training both a task classifier and a mixture-of-experts router, which involves computational overhead during the training phase. The method also requires access to responses from all candidate models on training data to compute quality scores, which may not be feasible in all deployment scenarios.

Fourth, our approach currently handles text-only prompts. It is a promising direction for future work to extend task discovery to multimodal inputs where task structure may emerge from visual or audio features.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

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

DeepSeek-AI. 2025a. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *arXiv preprint arXiv:2501.12948*.

DeepSeek-AI. 2025b. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

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

Aosong Feng, Balasubramaniam Srinivasan, Yun Zhou, Zhichao Xu, Kang Zhou, Sheng Guan, Yueyan Chen, Xian Wu, Ninad Kulkarni, Yi Zhang, Zhengyuan Shen, Dmitriy Bespalov, Soumya Smruti Mishra, Yifei Teng, Darren Yow-Bang Wang, Haibo Ding, and Lin Lee Cheong. 2025a. IPR: Intelligent prompt routing with user-controlled quality-cost trade-offs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 2484–2498.

Tao Feng, Yanzhen Shen, and Jiaxuan You. 2025b. Graphrouter: A graph-based router for LLM selections. In *The Thirteenth International Conference on Learning Representations*.

Tao Feng, Haozhen Zhang, Zijie Lei, Haodong Yue, Chongshan Lin, Ge Liu, and Jiaxuan You. 2025c. Llmrouter: An open-source library for llm routing. https://github.com/ulab-uiuc/LLMRouter. GitHub repository.

Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. 2021a. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021b. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021c. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Chris Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiacai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, Yang Liu, and Yahui Zhou. 2025. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. *arXiv preprint arXiv:2507.01352*.

AI @ Meta Llama Team. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2024. Routing to the expert: Efficient reward-guided ensemble of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1964–1974.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391.

NVIDIA. 2024. LLM Router: AI Blueprint. https://github.com/NVIDIA-AI-Blueprints/llm-router. GitHub repository.

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

OpenAI. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Qwen-Team. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158.

V. A. Traag, L. Waltman, and N. J. van Eck. 2019. From louvain to leiden: guaranteeing well-connected communities. *Sci Rep.*, 9(1):5233.

William Webber, Alistair Moffat, and Justin Zobel. 2010. A similarity measure for indefinite rankings. *ACM Trans. Inf. Syst.*, 28(4).

Dun Zhang, Jiacheng Li, Ziyang Zeng, and Fulong Wang. 2025a. Jasper and stella: distillation of sota embedding models. *arXiv preprint arXiv:2412.19048*.

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. 2025b. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Task Description Generation

Your task is to generate a short description of the task that a user's prompt intends to do.
You are allowed to perform chain-of-thought or thinking but the final answers should be in <task> tags with the following instructions:
- The description should be one short sentence.
- The description should cover different aspects of the tasks, including but not limited to task types (e.g., classification, generation, translation, ...), topic of interests (e.g., technology, health, finance, ...), domain of knowledge.
- Do not respond to the provided prompt. Your task is to generate a description of what it tries to do
- Do not include explanations, reasoning, context, or commentary of any kind in the task description
- Do not preface or conclude your answer with statements like "Based on my knowledge..." or "The task is..." in the task description
- Try to always output an answer
- Format your response exactly as follows:

<task> task description </task>

Now, please generate the task description for the following prompt: $\{p\}$

Figure 3: Prompt given to the LLM to generate task descriptions for offline task type discovery (Sect. 3.1.1).

A Prompt for Task Description Generation

Figure 3 shows the prompt we used to generate domain and task descriptions for training prompts during the offline task type discovery process in Section 3.1.1.

B Implementation Details

For Stage 1 task discovery, we use CLAUDE-SONNET-4.5 to generate task descriptions for each training prompt. We construct the prompt graph with $k = 5$ nearest neighbors based on embeddings from Sentence Transformer ALL-MINILM-L6-V2 (Reimers and Gurevych, 2019). We set the RBO threshold $\tau = 0.4$ and apply Leiden community detection for $\ell = 3$ iterations. The coverage threshold δ for adaptive candidate selection is set to 0.8.

The task type classifier is initialized from STELLA_EN_400M_V5 (Zhang et al., 2025a) and fine-tuned for 10 epochs with a learning rate of 2e-5 and per-device batch size of 8. For Stage 2, the prompt encoder is initialized from QWEN3-EMBEDDING-0.6B (Zhang et al., 2025b). The LLM embedding layer has dimension 512. Each quality estimation adapter consists of a 2-layer MLP with hidden dimensions 512. We first train the base model for 10 epochs, then fine-tune task-specific adapters for another 10 epochs while freezing other parts. All models use per-device batch size 8. All model training is done on 8 NVIDIA A100 GPUs. Training the full pipeline takes approximately 17 hours on 8 NVIDIA A100 GPUs (around 135 GPU-hours): 6 hours for the base model and 11 hours for task-specific adapter fine-tuning.

At inference, we set the aggregation weight $\alpha = 0.5$ in Eq. 5. We use SKYWORK-REWARD-V2-LLAMA-3.1-8B (Liu et al., 2025) as the reference quality function $Q^*(\cdot)$ for both training and evaluation.

C Task Classifier Analysis

The task classifier achieves 0.643 macro F1 on a held-out validation set of cluster assignments, which is a 332-class classification problem. On the test set, 71% of prompts (22,485 out of 31,774) are assigned to a discovered task type. Prompts with task assignments achieve a quality score of 0.665, compared to 0.619 for prompts in the "Others" category, confirming that task-specific routing provides meaningful improvements over the general fallback.