Task-Aware LLM Routing with Multi-Level Task-Profile-Guided Data Synthesis for Cold-Start Scenarios

Hui Liu[1], Bin Zou[2], Kecheng Chen[1], Jie Liu[1], Wenya Wang[3], Haoliang Li[1]

[1]City University of Hong Kong [2] University of Hong Kong [3]Nanyang Technological University

Abstract

Large language models (LLMs) exhibit substantial variability in performance and computational cost across tasks and queries, motivating routing systems that select models to meet user-specific cost–performance trade-offs. However, existing routers generalize poorly in cold-start scenarios where in-domain training data is unavailable. We address this limitation with a multi-level task-profile–guided data synthesis framework that constructs a hierarchical task taxonomy and produces diverse question–answer pairs to approximate the test-time query distribution. Building on this, we introduce TRouter, a task-type–aware router approach that models query-conditioned cost and performance via latent task-type variables, with prior regularization derived from the synthesized task taxonomy. This design enhances TRouter's routing utility under both cold-start and in-domain settings. Across multiple benchmarks, we show that our synthesis framework alleviates cold-start issues and that TRouter delivers effective LLM routing. The code is publicly available at this link[1].

1 Introduction

Large language models (LLMs) have been widely adopted by consumers and enterprises (Appel et al., 2025), yet their performance and cost vary substantially across tasks and queries due to parameter size and reasoning length differences (Hu et al., 2024). Moreover, users also have different preferences for performance and cost (Chen et al., 2023). For example, a startup deploying a customer chatbot may prioritize cost-effectiveness, whereas a research lab analyzing complex literature may favor high-performance models despite higher costs. This diversity motivates the LLM routing problem of assigning optimal models from a candidate pool to each query while aligning with user preferences (Feng et al., 2025; Lu et al., 2023).

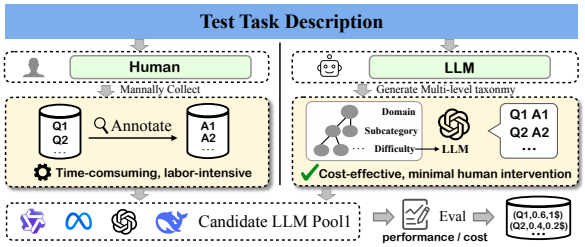

Figure 1: Comparison between the traditional data preparation pipeline and our proposed LLM-based data synthesis approach for the LLM router training.

As shown in Fig 1, most existing approaches assume access to training datasets (in-domain setting) with query-answer (QA) pairs and evaluate all candidate LLMs to obtain performance and cost metrics. They then train lightweight routers, either classifiers or regressors (Devlin et al., 2019) to select models. Classification-based approaches (Ong et al., 2024; Zhuang et al., 2024; Nguyen et al., 2024; Chen et al., 2024) label optimal LLMs according to user preferences and output the selected model during testing, while regression-based methods (Zhang et al., 2025b; Somerstep et al., 2025; Mohammadshahi et al., 2024; Šakota et al., 2024) predict performance and cost and select models via utility maximization, enabling flexible specifications such as batch routing (Mei et al., 2025).

However, real-world deployments frequently face cold-start scenarios without resources to collect labeled training data (Wang et al., 2025; Liu et al., 2025), especially for individual users and early-stage products. Moreover, pre-trained routers often generalize poorly at deployment, likely due to domain shifts between training and test-time user inputs (Zhang et al., 2025a). As shown in Table 3, both classification and regression-based methods (e.g., RouterDC and MetricRouter) exhibit limited robustness under cross-domain settings compared to a simple rule-based alternative (Adaptive LLM)

[1]https://github.com/less-and-less-bugs/ColdStartLLMRouter

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 22047–22076

that selects more powerful models as user cost tolerance increases. Additionally, while routing queries directly with separate LLMs may seem feasible, accurately characterizing each candidate model's capabilities remains difficult (Chang et al., 2024; Yehudai et al., 2025), making this approach unsuitable for cold-start scenarios.

To address the cold-start problem in LLM routing, we propose a multi-level task-profile-guided data synthesis framework that generates diverse question-answer pairs approximating the test-time query distribution. Drawing upon recent advances in LLM evaluation and deployment practices (Chang et al., 2024; Shao et al., 2024), we observe that both computational cost and model performance are intrinsically linked to task category and difficulty, further validated in Appendix C.3. Our framework employs a Task Type Generator that, when seeded with domain-relevant information, iteratively constructs a hierarchical task taxonomy spanning domain, subcategory, and difficulty dimensions. This hierarchical mechanism enables fine-grained control and enhanced sampling efficiency, while a Task Type Quality Evaluator ensures taxonomic cohesion and diversity throughout the generation process. By conceptualizing each difficulty-specific category as a distinct task profile, our Question-Answer Pair Generator produces non-redundant QA pairs for each profile, which candidate LLMs subsequently evaluate to generate routing training data for cold-start scenarios.

Departing from conventional approaches that directly model cost and performance from raw query features, we introduce TRouter, a task-type-aware LLM routing system that augments regression-based routing through the incorporation of latent task-type variables and employs a prior distribution over the synthesized task taxonomy as a regularization. TRouter effectively captures latent task semantic structures that transcend surface-level query characteristics, achieving superior routing performance and enhanced robustness across cold-start and in-domain configurations. Extensive experiments across multiple LLM pools and evaluation protocols show that our synthesis framework mitigates cold-start limitations, while TRouter exploits task-specific information to yield substantial improvements in routing utility. The contribution of this work is as follows:

- To the best of our knowledge, we are the first to identify the cold-start problem in LLM routing and introduce a multi-level task-profile–guided data synthesis framework that approximates the test-time query distribution.
- We propose TRouter, a task-type–aware router that integrates latent task-type variables and a prior over the synthesized task taxonomy into the regression-based paradigm, improving routing utility.
- We validate the effectiveness of the synthesis framework and TRouter across multiple LLM pools and evaluation protocols in both cold-start and in-domain settings

2 RELATED WORKS

LLM Routing. LLM routing aims to optimize the allocation of candidate LLMs for input queries under user-defined preferences. While early work focuses on performance maximization (Jiang et al., 2023; Shnitzer et al., 2023), recent approaches emphasize performance-cost trade-offs as model capabilities improve and call costs rise (Zhang et al., 2023; Ding et al., 2024). Using two main paradigms, these methods evaluate candidate LLMs on training queries and train a small model, like BERT, as a router. Classification-based approaches (Chen et al., 2024; Srivatsa et al., 2024) annotate optimal LLMs for training queries and learn to predict the best model. For example, GraphRouter (Feng et al., 2025) formulates routing as edge classification over heterogeneous graphs modeling query-model relationships. Regression-based methods (Somerstep et al., 2025; Mei et al., 2025) predict performance and cost metrics for query-model pairs and then optimize user-defined utility functions at inference. Our work adopts the regression paradigm because it can enable batch routing with model quotas (Cheng et al., 2024) and support fine-grained preference control, whereas classification methods typically operate per-query. Additionally, alternative approaches directly fine-tune LLMs for model selection, potentially achieving higher accuracy but increasing inference costs and latency (Zhang et al., 2025a,b; Aggarwal et al., 2023). At the same time, cascade-based methods assign fixed LLM sequences to the whole test set, limiting their performance in query-level routing (Chen et al., 2023; Dai et al., 2024).

Data Synthesis using LLMs. LLM-based data synthesis has substantially reduced annotation costs, enabling scalable construction of large-scale

training datasets. Recent work (Tan et al., 2024) identifies three core principles for effective synthesis: diversity, achieved via conditional prompting and multi-step generation to broaden topical and stylistic coverage; quality, maintained through filtering, label refinement, and cross-consistency checks to limit hallucinations and noise; and domain specificity, incorporating external knowledge and task-tailored prompts to inject expertise. However, empirical studies (Yang et al., 2025; Shumailov et al., 2024) demonstrate that synthesis pipelines are highly customized, with different applications requiring distinct approaches. To our knowledge, no existing work has explored data synthesis methods to approximate test-time query distributions for LLM routing training.

3 METHODOLOGY

To mitigate cold-start challenges in practical LLM routing, we propose a multi-level task-profile–guided data synthesis framework that generates diverse question–answer pairs to approximate realistic query distributions in Sec. 3.2. Leveraging the synthesized task taxonomy, we further introduce TRouter, a task-type–aware router that treats task type as a latent variable to predict performance and cost in Sec. 3.3. Fig. 2 provides an overview of our proposed data synthesis strategy and TRouter.

3.1 Problem formulation

Let $\mathcal{M} = \{m_1, \ldots, m_M\}$ denote a set of M candidate LLMs. Following Feng et al. (2025), for a user query q, LLM routing aims to select a model $m^* \in \mathcal{M}$ to maximize a utility function as below:

$$m^* = \arg\max_{m \in \mathcal{M}} U(m, q) = \mu_r \cdot \mathbf{r}(m, q) - \mu_c \cdot \mathbf{c}(m, q)$$
$$\text{s.t.} \quad \mu_r + \mu_c = 1, \quad \mu_r, \mu_c \geq 0 \tag{1}$$

where $\mathbf{r}(\cdot)$ and $\mathbf{c}(\cdot)$ are learnable performance and cost functions, and μ_r, μ_c represent user preferences over the performance–cost trade-off. Performance evaluation may vary depending on task types (e.g., accuracy and F1), while cost is determined by input/output token numbers and model-specific per-token pricing. In a given test scenario with test set $\mathcal{D}_{\text{test}}$, prior work often assumes access to an in-domain labeled training set and evaluates all models in $\mathcal{M}$ to collect tuples $\{q_i, c_{i,j}, r_{i,j}, m_j\}$, where $c_{i,j}$ and $r_{i,j}$ are the observed cost and performance of model m_j on query q_i. These tuples constitute the final router training data $\mathcal{D}_{\text{train}}$.

3.2 Task-profile-guided Data Synthesis

As obtaining sufficient manually annotated data is challenging in the cold-start setting, we propose a multi-level task-profile-guided data synthesis framework, comprising task type generator, task type quality evaluator, and question-answer pair generator. Each component is detailed as follows.

Task Type Generator. We adopt a three-level task taxonomy from general to fine-grained, comprising domain, subcategory, and difficulty. A short name, a definition, and an illustrative example are generated to specify each task type. Beginning with manually authored domain descriptions, this generator recursively expands the taxonomy by producing multiple distinct, non-overlapping child types per step. Parent-type definitions are used as conditional prompts to promote diversity and relevance. To avoid excessive granularity, we enforce maximum node counts via prompt constraints.

Task Type Quality Evaluator. After the task type generator proposes a set of candidate child task types for a given parent type, the task type quality evaluator conducts a self-critique (Shao et al., 2024) against predefined criteria, such as minimal redundancy, specificity, and completeness. The evaluator first shuffles the current set and ingests it as input, then decides whether revisions are warranted. If so, it iteratively refines the set. This process repeats until the evaluator determines that no further modifications are necessary across multiple consecutive assessment rounds. At that point, the candidate set is finalized as the child task type set for the corresponding parent type.

Question-Answer Pair Generator. After constructing the hierarchical task taxonomy, we use the short name and description of each difficulty-level task type, together with the descriptions of its parent task types, to form a task profile. For each profile, our objective is to generate N_d QA pairs. The QA pair generator produces QA pairs in batches and applies a sentence-transformer model to compute the semantic similarity between newly generated pairs and the existing collection. Any QA pair whose maximum similarity to an existing pair exceeds 0.9 is discarded to filter out near-duplicate ones. This batched generation and filtering process is repeated iteratively until N_d diverse QA pairs are obtained for each task profile.

Overall, our proposed data synthesis framework requires only domain descriptions as input to guide

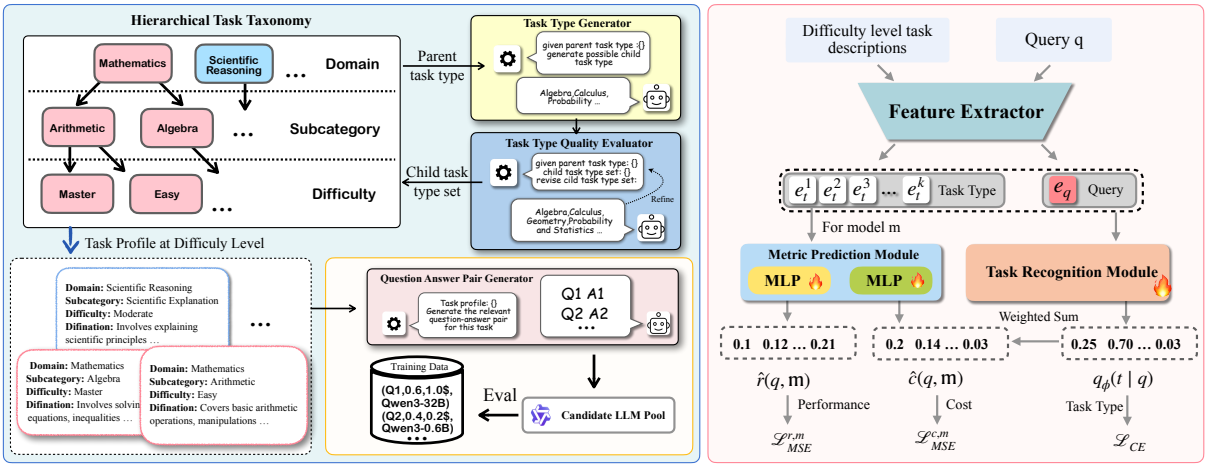

Figure 2: (a) Overview of our proposed task-profile-guided data synthesis framework. The task type generator and quality evaluator collaboratively construct a hierarchical task taxonomy, while the question-answer pair generator produces diverse QA pairs based on task profiles from difficulty-level task types. (b) Overview of our proposed task-type–aware LLM routing method, introducing a latent task-type variable into cost and performance prediction.

LLMs in constructing a synthetic QA dataset, thus obviating manual data collection and annotation. Similar to $\mathcal{D}_{\text{train}}$, all candidate LLMs in $\mathcal{M}$ are evaluated on the QA dataset to collect routing training data, denoted as $\mathcal{D}_{\text{syn}}$ in a cold-start setting.

3.3 Task-type–aware Routing Framework

3.3.1 Theoretical Analysis

Let $h \in \mathcal{H} = \{r, c\}$ denote an evaluation metric, where r and c represent performance and cost, respectively. In contrast to prior routing methods that directly model the conditional distributions of cost and performance for query-model pairs, i.e., $p(h \mid q, m)$, TRouter introduces an unobserved latent task variable $t \in \mathcal{T}$, where $\mathcal{T}$ represents the task-type space. The conditional distribution is then decomposed as:

$$p(h \mid q, m) = \sum_{h \in \mathcal{T}} p(h \mid q, m, t) p(t \mid q, m). \tag{2}$$

By leveraging t as the intermediate representation across different metrics, TRouter disentangles the impact of task semantics from other input features.

To simplify the probability distribution, we adopt two conditional independence assumptions: (i) performance is independent of the query given the task type and model, and (ii) task type is independent of the model given the query, which are denoted as:

$$p(h \mid t, q, m) = p(h \mid t, m), \quad p(t \mid q, m) = p(t \mid q). \tag{3}$$

Under these assumptions, the conditional distribution $p(h \mid q, m)$ can be reformulated as:

$$p(h \mid q, m) = \sum_{t \in \mathcal{T}} p(h \mid t, m) p(t \mid q). \tag{4}$$

Intuitively, the query q induces a posterior $p(t \mid q)$ over task types, and each task type t then specifies a task-model-pair dependent metric conditional distribution $p(h \mid t, m)$.

To train the metric predictor $p_\theta(h \mid q, m)$, given a training corpus $\mathcal{D}$ (e.g., $\mathcal{D}_{\text{syn}}$ and $\mathcal{D}_{\text{train}}$), we maximize the log-likelihood through:

$$\begin{aligned} \mathcal{L} &= \sum_{(q,h,m) \in \mathcal{D}} \log p_\theta(h \mid q, m) \\ &= \sum_{(q,h,m) \in \mathcal{D}} \log \sum_{t \in \mathcal{T}} p_\theta(h \mid t, m) p(t \mid q), \end{aligned} \tag{5}$$

where $p(t \mid q)$ is the prior over task types and θ denotes learnable parameters. However, directly maximizing such a likelihood is difficult due to marginalization over the latent t and because t is unobserved at deployment. We therefore introduce a variational posterior $q_\phi(t \mid q)$ to approximate $p(t \mid q)$ and optimize the evidence lower bound (ELBO) (Kingma and Welling, 2013) as follows

$$\begin{aligned} \log p_\theta(h \mid q, m) \geq\ & \mathbb{E}_{q_\phi(t \mid q)} \left[\log p_\theta(h \mid t, m) \right] \\ & - \mathrm{KL}\left(q_\phi(t \mid q) \,\|\, p(t \mid q) \right), \end{aligned} \tag{6}$$

where ϕ denotes learnable parameters. The first term is a reconstruction term that encourages task representations predictive of metric h for model m; the second term regularizes the variational posterior toward the prior, improving calibration and mitigating overfitting. A complete derivation of Eq (6) is detailed in Appendix B.

3.3.2 Model Implementation

We instantiate our probabilistic framework with a task recognition module ($q_\phi(t \mid q)$) and multiple metric prediction modules ($p_\theta(h \mid t, m)$). We define the full set of task types at the difficulty level of the task taxonomy introduced in Sec. 3.2 as $\mathcal{T}$ with $|\mathcal{T}| = K$. For a given query q, while it may belong to multiple task types in $\mathcal{T}$, we assign a single ground-truth label by selecting the task type corresponding to the query's annotated difficulty level. This label is represented as a one-hot vector $y \in \mathbb{R}^K$ and serves as prior ($p(t \mid q)$) for the recognition module. Both the query q and the textual definition of task types in $\mathcal{T}$ are encoded using a pretrained sentence encoder. The resulting representations are then passed through two-layer multilayer perceptrons (MLPs) to produce a query embedding e_q and task-type embeddings $\{e_t^i\}$ where $i \in [1, K]$.

Task recognition module. For the query q, we concatenate its embedding and all task embeddings $\{e_t^i\}$ and pass them into an MLP, followed by a softmax function with a temperature parameter τ to obtain the predicted task type distribution $l \in \mathbb{R}^K$ as $q_\phi(t \mid q)$. Then we train this module using cross-entropy loss over the training set $\mathcal{D}$ as follows:

$$\mathcal{L}_{\text{CE}} = -\sum_{q \in \mathcal{D}} y \log q_\phi(t \mid q), \tag{7}$$

enforcing the optimization of KL-term in Eq. (6).

Metric prediction module. For each pair of metric h and model $m \in \mathcal{M}$, we predict the metric by computing the weighted combination of task-specific predictions using the predicted task distribution l from the recognition module as follows:

$$\hat{h}(q, M) = \sum_{i \in [1,K]} l_i \cdot \sigma\left(\text{MLP}_{h,m}(e_t^i)\right), \tag{8}$$

where σ denotes the sigmoid function, ensuring normalized outputs, and l_i represents the possibility that q belongs to i-th task type. We train this module with MSE loss between predicted and observed metric $h(q, M)$ as follows:

$$\mathcal{L}_{\text{MSE}}^{h,m} = \sum_{q \in \mathcal{D}} \left(\hat{h}(q, m) - h(q, m)\right)^2, \tag{9}$$

which serves as reconstruction term in Eq. (6). Then, we combine both losses to form the final training objective over the training set $\mathcal{D}$ as follows:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \frac{1}{|\mathcal{M}||\mathcal{H}|} \sum_{m \in \mathcal{M}} \sum_{h \in \mathcal{H}} \mathcal{L}_{\text{MSE}}^{h,m}. \tag{10}$$

Table 1: Overview of Datasets.

Dataset	Task Type	Metric	Cases
Alpaca (Taori et al., 2023)	Hybrid QA	F1	2000
GSM8K (Cobbe et al., 2021)	Reasoning	Accuracy	2000
SQuAD (Rajpurkar et al., 2016)	Reading Comprehension	F1	2000
Multi-News (Fabbri et al., 2019)	Summary	F1	2000

Table 2: Overview of open-source and commercial LLMs, their sizes, and costs per million tokens.

Model	Size	Input (¥/M)	Output (¥/M)
Open-source			
Qwen3-235B-A22B	235B	2.00	8.00
Qwen3-32B	32B	2.00	8.00
Qwen3-14B	14B	1.00	4.00
Qwen3-8B	8B	0.50	2.00
Qwen3-1.7B	1.7B	0.30	1.20
Qwen3-0.6B	0.6B	0.30	1.20
Commercial			
Gemini-2.5-Flash	–	2.14	17.86
Gemini-2.5-Flash-Lite	–	0.71	2.86
Gemini-2.0-Flash	–	0.71	2.86
Gemini-2.0-Flash-Lite-Preview	–	0.54	2.14
Doubao-Seed-1.6-Flash	–	0.15	1.50

During inference, for each query, we compute predicted metrics for each model via marginalization over the task space $\mathcal{T}$ and select the model that maximizes the utility function $U(m, q)$ in Eq. (1).

4 Experiments

4.1 Experimental Setting

Datasets and Candidate LLMs. Following recent work (Feng et al., 2025), we select data from four distinct tasks, with their statistics in Table 1. We further validate the versatility of our approach on four additional tasks in Appendix C.2. Instead of using outdated LLMs, to better reflect realistic deployment demands, we employ more advanced Qwen 3 models of varying sizes in our primary experiments and conduct supplementary experiments using a group of proprietary models in Table 2.

Data Prepossessing. We aggregate all sampled data from the four tasks. For each query, we utilize the candidate LLMs in Table 2 to generate corresponding responses. While prior work often assumes that the evaluation protocol for the test scenario is available, we argue that such assumptions do not always hold in real-world settings. Therefore, to evaluate response quality, in addition to employing traditional metrics in Table 2, we apply an LLM-as-a-judge framework (Gu et al., 2024), where a designated LLM evaluates responses based on predefined rules, serving as a proxy for ground-truth performance. Moreover, the cost of each model is computed by summing the number of input and output tokens, weighted by the corresponding price. Upon constructing the multi-

task interaction dataset, we divide it into training, validation, and test subsets using a 7:1:2 ratio.

For the synthetic data generated via the task-profile guided data synthesis method (Sec. 3.2), we estimate performance using the LLM-as-a-judge and calculate cost following the same procedure.

Metrics. Since users may prioritize performance and cost differently, we define three user preference scenarios, including Cost First, Balanced, and Performance First. Specifically, we assign weights (μ_r, μ_c) of $(0.2, 0.8)$, $(0.5, 0.5)$, and $(0.8, 0.2)$ to these scenarios, respectively. We avoid setting $(1, 0)$ for Performance First to prevent degenerate routing, always selecting the largest LLM. To enable utility-based optimization, following (Feng et al., 2025), we normalize performance and cost metrics to remove scale discrepancies.

Baseline Methods. To evaluate the effectiveness of our proposed TRouter, we compare it against two categories of baseline methods. The first category, Cold-Start Baselines, includes approaches that do not rely on in-domain training data $\mathcal{D}_{\text{train}}$: (1) Smallest LLM, always selecting the smallest available model to minimize inference cost; (2) Largest LLM, selecting the largest model to maximize performance; (3) Adaptive LLM, dynamically selecting more expensive model based on the user's tolerance for costs; and (4) Prompt LLM, utilizing an external LLM to select a candidate model based on a prompt that encodes the input query, candidate model pool, and user objectives. In this setting, TRouter is trained exclusively on synthetic data $\mathcal{D}_{\text{syn}}$ designed to simulate realistic task scenarios.

The second category, In-Domain Baselines, includes methods that assume access to $\mathcal{D}_{\text{train}}$: (5) RouterDC (Chen et al., 2024), a dual-contrastive learning-based selector; (6) GraphRouter (Feng et al., 2025), which models routing as edge prediction on a heterogeneous graph of tasks, queries, and models; (7) MetricRouter (Mei et al., 2025), a regression-based method that predicts performance and cost using sentence embeddings; (8) FrugalGPT (Chen et al., 2023), which estimates generation quality under cost constraints; (9) C2MAB-V (Dai et al., 2024), a contextual bandit approach for adaptive model selection; and (10) Oracle, a theoretical upper bound that selects the model with the highest utility using ground-truth cost and performance. In the in-domain setting, TRouter is also trained on $\mathcal{D}_{\text{train}}$ after annotating task type using LLMs. Additionally, we adapt RouterDC and MetricRouter to the cold-start setting using cross-dataset training for the cold-start comparisons.

Implementation Details. We instantiate the data synthesis pipeline with gpt-4.1 and validate versatility by substituting gemini-2.5-flash. Performance is estimated via an LLM-as-a-judge using gpt-4.1-nano for cost efficiency (Table 4). Starting from six seed domains, we expand to ten (Table 17); per domain, we generate up to ten subcategories and five difficulty levels (Tables 18, 19). The task type quality evaluator terminates after three consecutive no-change rounds (Table 20). The QA pair generator produces 40 QA pairs per task profile (batch size 8). The resulting datasets comprise, for gpt-4.1: 10 domains, 103 subcategories, 447 difficulty nodes, and 17,880 QA pairs; and for gemini-2.5-flash: 10 domains, 98 subcategories, 380 difficulty nodes, and 15,200 QA pairs.

For the proposed TRouter, we operate at the difficulty level for $\mathcal{T}$, encoding queries and task-type descriptions with all-MiniLM-L6-v2 [2] and mapping them into 256-dimensional vectors. Models are trained with learning rate 1e-4. We use 30 training and 10 validation QA pairs per task type in the cold-start setting. Due to page limitations, full Experimental Settings are provided in Appendix A.

4.2 Main Experimental Results

We evaluate our proposed task-profile guided data strategy and TRouter in the cold-start setting, where training data $\mathcal{D}_{\text{train}}$ from the test scenario is unavailable, and the in-domain setting, where such data can be collected. We report results under two evaluation protocols, including traditional metrics and LLM-as-a-judge in Table 3 and Table 4. The results lead to three main conclusions.

First, the task-profile guided data strategy effectively mitigates the inability of LLM routers to adapt under cold-start settings. For example, TRouter trained on the synthetic dataset $\mathcal{D}_{\text{syn}}$ achieves substantial gains over the best baselines that do not rely on $\mathcal{D}_{\text{train}}$ (i.e., Adaptive LLM) across different evaluation protocols. When using LLM-as-a-judge in Table 4, TRouter, trained solely on $\mathcal{D}_{\text{syn}}$, surpasses most in-domain baselines. However, methods relying on cross-domain training (RouterDC and MetricRouter) suffer from severe performance degradation due to the domain shift in the distribution of queries and corresponding

[2] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Table 3: Results across three user preference settings in both cold-start and in-domain scenarios. For in-domain training, candidate LLM performance is obtained using **traditional metrics**. Bold and underline indicate the best and second-best results under each user preference, respectively. We use $\star$ to denote routers trained on any three test tasks and evaluated on the remaining tasks (cross-domain). The symbols ▲ and • indicate that our data synthesis framework is derived using gpt-4.1 and gemini-2.5-flash, respectively.

		Cost-first			Balance			Performance-first			
Scenario	Method	Cost	Performance	Utility	Cost	Performance	Utility	Cost	Performance	Utility	Utility Sum
Cold-start	Smallest LLM	0.0230	0.2004	0.0217	0.0230	0.2004	0.0887	0.0230	0.2004	0.1557	0.2661
	Largest LLM	0.3098	0.4383	−0.1601	0.3098	0.4383	0.0643	0.3098	0.4383	0.2887	0.1928
	Adaptive LLM	0.0230	0.2004	0.0217	0.0641	0.4260	0.1809	0.3098	0.4383	0.2887	0.4913
	Prompt LLM	0.0427	0.2131	0.0085	0.1128	0.3996	0.1434	0.3098	0.4383	0.2887	0.4406
	RouterDC*	0.0391	0.2548	0.0197	0.0599	0.3580	0.1490	0.0650	0.3899	0.2989	0.4676
	MetricRouter*	0.0325	0.1916	0.0123	0.0446	0.2027	0.0703	0.1012	0.2118	0.1492	0.2319
	Ours$^▲$	0.0621	0.4257	**0.0355**	0.0641	0.4262	**0.1811**	0.1721	0.4315	<u>0.3108</u>	<u>0.5274</u>
	Ours$^•$	0.0327	0.3068	<u>0.0352</u>	0.0526	0.4144	<u>0.1809</u>	0.0701	0.4201	**0.3221**	**0.5382**
In-domain	RouterDC	0.0578	0.4367	0.0411	0.0593	0.4290	0.1849	0.0660	0.4425	0.3408	0.5667
	GraphRouter	0.0480	0.3059	0.0228	0.0783	0.3976	0.1597	0.0628	0.4113	0.3165	0.4989
	FrugalGPT	0.0230	0.2004	0.0217	0.0890	0.4141	0.1625	0.3535	0.4267	0.2706	0.4548
	C2MAB-V	0.0640	0.4213	0.0323	0.0635	0.4127	0.1746	0.0758	0.4146	0.3166	0.5234
	MetricRouter	0.0479	0.4125	0.0442	0.0559	0.4382	0.1911	0.0957	0.4474	<u>0.3388</u>	0.5741
	Ours$^▲$	0.0393	0.4165	**0.0518**	0.0477	0.4424	**0.1974**	0.0751	0.4479	**0.3433**	**0.5925**
	Ours$^•$	0.0400	0.4127	<u>0.0509</u>	0.0481	0.4353	<u>0.1936</u>	0.0876	0.4389	0.3336	<u>0.5836</u>
Oracle		0.0354	0.4943	0.0705	0.0417	0.5081	0.2332	0.0513	0.5127	0.3999	0.7037

Table 4: Results across three user preference settings in both cold-start and in-domain scenarios. For in-domain training, candidate LLM performance is obtained using **LLM-as-a-judge**. Bold and underline indicate the best and second-best results under each user preference, respectively. We use $\star$ to denote routers trained on any three test tasks and evaluated on the remaining tasks (cross-domain).

		Cost-first			Balance			Performance-first			
Scenario	Method	Cost	Performance	Utility	Cost	Performance	Utility	Cost	Performance	Utility	Utility Sum
Cold-start	Smallest LLM	0.0226	0.3342	0.0488	0.0226	0.3342	0.1558	0.0226	0.3342	0.2629	0.4675
	Largest LLM	0.3082	0.5158	−0.1434	0.3082	0.5158	0.1038	0.3082	0.5158	<u>0.3510</u>	0.3113
	Adaptive LLM	0.0226	0.3342	0.0488	0.0637	0.5024	<u>0.2194</u>	0.3082	0.5158	<u>0.3510</u>	<u>0.6191</u>
	Prompt LLM	0.0394	0.4053	0.0495	0.1088	0.4853	0.1883	0.3082	0.5158	<u>0.3510</u>	0.5888
	RouterDC*	0.0713	0.5573	**0.0544**	0.0914	0.5231	0.2159	0.1148	0.5297	0.2074	0.4777
	MetricRouter*	0.0620	0.4967	0.0498	0.1240	0.5397	0.2078	0.1833	0.5493	0.1830	0.4406
	Ours	0.0617	0.4992	<u>0.0505</u>	0.0637	0.5030	**0.2196**	0.0740	0.4997	**0.3850**	**0.6551**
In-domain	RouterDC	0.0369	0.4387	<u>0.0582</u>	0.0572	0.4577	0.2002	0.0638	0.4497	0.3470	0.6054
	GraphRouter	0.0240	0.3395	0.0487	0.0316	0.3616	0.1650	0.0337	0.3698	0.2891	0.5028
	FrugalGPT	0.0230	0.3302	0.0476	0.0899	0.3914	0.1508	0.1464	0.4872	0.3605	0.5589
	C2MAB-V	0.0636	0.5003	0.0492	0.0637	0.5024	<u>0.2194</u>	0.1290	0.5054	<u>0.3785</u>	0.6471
	MetricRouter	0.0376	0.4526	**0.0604**	0.0647	0.5024	0.2189	0.1106	0.4988	0.3769	<u>0.6512</u>
	Ours	0.0421	0.4705	**0.0604**	0.0549	0.4982	**0.2223**	0.1080	0.5152	**0.3906**	**0.6733**
Oracle		0.0388	0.6390	0.0968	0.0633	0.6984	0.3175	0.0690	0.7013	0.5472	0.9615

metrics across different tasks, with the regression-based MetricRouter being particularly vulnerable. These findings indicate that LLM-driven data synthesis, conditioned only on domain descriptions, can generate effective supervision signals for routing, thus enabling cold-start routing in resource-constrained scenarios.

Second, our proposed TRouter demonstrates advantages in both cold-start and in-domain settings. For instance, in the in-domain scenario, TRouter consistently outperforms existing learning-based routing frameworks across diverse user preferences, whereas competing baselines (e.g., C2MAB-V) often exhibit large performance variance. This robustness possibly stems from TRouter's task-type guided regularization, which injects structured task semantics beyond query semantics.

Third, although synthetic data is a strong choice for cold-start, TRouter still benefits from training on data drawn from the same distribution as the test scenario. As shown in Table 3, the TRouter in the in-domain setting improves the utility sum by more than 0.05 relative to the cold-start counterpart, while this utility gap narrows considerably under LLM-as-a-judge in Table 4. We suggest that this is due to differing performance distribution of candidate LLMs induced by different evaluation protocols, as corroborated by Fig. 5 in the appendix. Consequently, for practical deployment, we recommend that end-users and router providers adopt aligned evaluation protocols, e.g., using LLM-as-a-judge consistently when deploying routers.

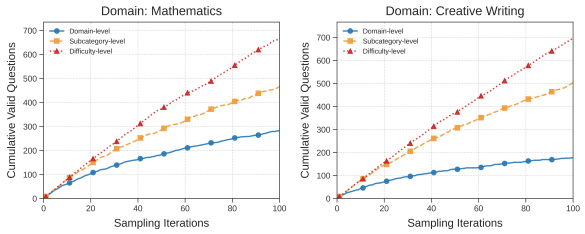

Figure 3: Effect of using task types of different taxonomy level to construct task-profile on sampling efficiency on Mathematics and Creative Writing domains.

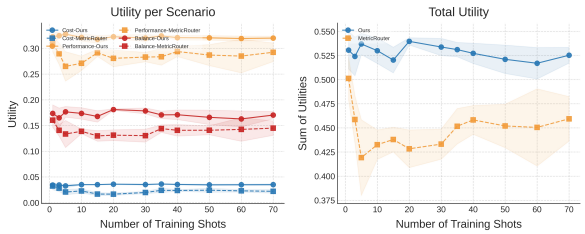

Figure 4: Effect of shot number in the cold-start setting for TRouter and MetricRouter.

4.3 Ablation Study

Effect of task profiles across taxonomy levels. We adopt a hierarchical task taxonomy, including domain, subcategory, and difficulty, in our data synthesis framework. In the question–answer pair generator, we instantiate a task profile using the difficulty-level task type as a conditional prompt to guide QA pair generation. To assess the impact of different taxonomy levels on sampling efficiency, we compare this approach with constructing task profiles at the domain and subcategory levels in Fig. 3. The results show that under a fixed number of sampling iterations, difficulty-conditioned profiles produce an almost linear increase in validated QA pairs, whereas domain and subcategory-conditioned profiles exhibit slowing gains due to accumulating redundancy across two representative domains. These results indicate that difficulty-level conditioning yields more diverse samples and better approximates the test-time query distribution in the cold-start routing setting.

Effect of shot number in cold-start setting. We train TRouter on the synthetic dataset $\mathcal{D}_{\text{syn}}$ with varying shots per task in Fig. 4. Although our main experiments use 30 shots, performance with five or fewer shots is already near-optimal across diverse user preferences in LLM routing. This demonstrates that TRouter is sample-efficient and can be deployed cheaply in the cold-start setting.

Table 5: Effect of $\mathcal{T}$ across three taxonomy levels. Bold denotes the first best values under each user preferences.

Method	Cost	Balance	Performance	Utility Sum
Cold Start				
Ours w/ domain regulation	0.0343	0.1809	**0.3250**	**0.5402**
Ours w/ subcategory regulation	0.0350	0.1740	0.3204	0.5294
Ours	**0.0355**	**0.1811**	0.3108	0.5274
In-domain				
Ours w/ domain regulation	0.0505	0.1887	0.3249	0.5641
Ours w/ subcategory regulation	0.0510	0.1920	0.3379	0.5809
Ours	**0.0518**	**0.1974**	**0.3433**	**0.5925**

Effect of task regularization in cold-start setting. Removing the task regularization in Eq. (6), TRouter degrade to MetricRouter. As shown in Fig. 4, when trained on the same $\mathcal{D}_{\text{syn}}$, TRouter consistently outperforms MetricRouter across all shot numbers. This advantage stems from incorporating a task-type prior, which enables TRouter to handle numerous task types and better cope with complex scenarios. We also observe that MetricRouter's performance is unstable, which decreases with more shots initially but increases thereafter, while TRouter remains robust in increasing shot number, further validating its effectiveness.

Effect of $\mathcal{T}$ across taxonomy levels. We instantiate TRouter with task types $\mathcal{T}$ drawn from the difficulty level of our taxonomy in the main experiments. In Table 5, we compare alternatives that use domain and subcategory-level types. The results show that in in-domain setting, difficulty-based $\mathcal{T}$ yields the best performance, while domain-based $\mathcal{T}$ in cold-start settings, likely because finer-grained difficulty types introduce interference when task-type priors are not yet calibrated. Moreover, after in-domain adaptation, the difficulty-level $\mathcal{T}$ benefits from its larger, more granular type space, which provides richer priors beyond query semantics, enabling more accurate routing.

5 Conclusions

We address the cold-start challenge in LLM routing by introducing a multi-level task-profile–guided data synthesis framework and TRouter, a task-type–aware LLM router. Our synthesis approach constructs diverse, taxonomy-aligned QA data that approximates test-time query distribution without requiring in-domain datasets. TRouter augments regression-based routing with latent task-type variables and a prior over the synthesized taxonomy, capturing structure beyond surface semantics. Across experiments on multiple LLM pools and evaluation protocols, we demonstrate our syn-

thesis framework enables effective cold-start training, and TRouter consistently improves routing utility in both cold-start and in-domain settings.

Limitations

We discussed three imitations of our work. First, the task-profile–guided synthesis requires minimal human input to specify candidate domains (or a short application description from which the LLM can derive domains). This introduces unavoidable human intervention. In practical deployments (e.g., customer support), however, providers typically know the high-level context and can seed domains manually or automatically generate them using LLM. Compared with existing routers that generally require in-domain labeled data, our approach is better suited to cold-start settings. Moreover, allowing limited human input provides desirable control and flexibility, such as pruning low-quality child task types and injecting human priors to refine our three-level taxonomy to an application-specific taxonomy. Second, our synthesized QA pairs are not exhaustively validated. Prior work on data synthesis for training LLMs often relies on strict rule-based filtering to avoid performance collapse. In contrast, our setting models cost–performance relationships at the task-type level, i.e., across distributions of queries rather than individual QA pairs, making TRouter more robust to noise in any single QA pair and reducing the need for heavy-handed data quality control. The empirical results in Fig. 4 also support this hypothesis. Third, using LLM-as-a-Judge may introduce model bias. In our main experiments, we mitigate this by selecting `gpt-4.1-nano-2025-04-14` as the judge model while evaluating candidate models from the Qwen and Gemma families, reducing the risk of self-preference bias. Notably, LLM-as-a-Judge aligns more closely with human preferences than traditional metrics. For instance, in tasks like GSM8K, conventional metrics may misjudge performance due to formatting issues, whereas LLM-based evaluation better captures semantic correctness. As discussed in Sec 4.2 (third point), consistency between the evaluation setup and the test scenario is crucial. In practice, we recommend aligning the evaluation protocol with the target use case and incorporating additional cross-checking mechanisms for robustness.

Acknowledgments

We thank the UGC FITE (6460001), DOMG (9229193), and NTU Start-Up Grant (#023284-00001), Singapore.

References

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, and 1 others. 2023. Automix: Automatically mixing language models. *arXiv preprint arXiv:2310.12963*.

Ruth Appel, Peter McCrory, Alex Tamkin, Michael Stern, Miles McCain, and Tyler Neylon. 2025. Anthropic economic index report: Uneven geographic and enterprise ai adoption.

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*.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, and 1 others. 2024. A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, 15(3):1–45.

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

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

Ke Cheng, Wen Hu, Zhi Wang, Peng Du, Jianguo Li, and Sheng Zhang. 2024. Enabling efficient batch serving for lmaas via generation length prediction. In *2024 IEEE International Conference on Web Services (ICWS)*, pages 853–864. IEEE.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Xiangxiang Dai, Jin Li, Xutong Liu, Anqi Yu, and John Lui. 2024. Cost-effective online multi-llm selection with versatile reward models. *arXiv preprint arXiv:2405.16587*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. 2024. Hybrid llm: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618*.

Alexander R Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R Radev. 2019. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. *arXiv preprint arXiv:1906.01749*.

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

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, and 1 others. 2024. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*.

Neel Guha, Julian Nyarko, Daniel Ho, Christopher Ré, Adam Chilton, Alex Chohlas-Wood, Austin Peters, Brandon Waldon, Daniel Rockmore, Diego Zambrano, and 1 others. 2023. Legalbench: A collaboratively built benchmark for measuring legal reasoning in large language models. *Advances in neural information processing systems*, 36:44123–44279.

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

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*.

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Tom Kocmi, Eleftherios Avramidis, Rachel Bawden, Ondřej Bojar, Anton Dvorkovich, Christian Federmann, Mark Fishel, Markus Freitag, Thamme Gowda, Roman Grundkiewicz, Barry Haddow, Philipp Koehn, Benjamin Marie, Christof Monz, Makoto Morishita, Kenton Murray, Masaaki Nagata, Toshiaki Nakazawa, Martin Popel, and 3 others. 2023. Findings of the 2023 conference on machine translation (WMT23): LLMs are here but not quite there yet. In *Proceedings of the Eighth Conference on Machine Translation*, pages 1–42, Singapore. Association for Computational Linguistics.

Hui Liu, Wenya Wang, Kecheng Chen, Jie Liu, Yibing Liu, Tiexin Qin, Peisong He, Xinghao Jiang, and Haoliang Li. 2025. Enhancing zero-shot image recognition in vision-language models through human-like concept guidance. *arXiv preprint arXiv:2503.15886*.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2023. Routing to the expert: Efficient reward-guided ensemble of large language models. *arXiv preprint arXiv:2311.08692*.

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

Alireza Mohammadshahi, Arshad Rafiq Shaikh, and Majid Yazdani. 2024. Routoo: Learning to route to large language models effectively. *arXiv preprint arXiv:2401.13979*.

Quang H Nguyen, Thinh Dao, Duy C Hoang, Juliette Decugis, Saurav Manchanda, Nitesh V Chawla, and Khoa D Doan. 2024. Metallm: A high-performant and cost-efficient dynamic framework for wrapping llms. *arXiv preprint arXiv:2407.10834*.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. 2024. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. 2022. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 248–260. PMLR.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Marija Šakota, Maxime Peyrard, and Robert West. 2024. Fly-swat or cannon? cost-effective language model choice via meta-modeling. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 606–615.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

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

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. 2024. Ai models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759.

Seamus Somerstep, Felipe Maia Polo, Allysson Flavio Melo de Oliveira, Prattyush Mangal, Mírian Silva, Onkar Bhardwaj, Mikhail Yurochkin, and Subha Maity. 2025. Carrot: A cost aware rate optimal router. *arXiv preprint arXiv:2502.03261*.

KV Srivatsa, Kaushal Kumar Maurya, and Ekaterina Kochmar. 2024. Harnessing the power of multiple minds: Lessons learned from llm routing. *arXiv preprint arXiv:2405.00467*.

Zhen Tan, Dawei Li, Song Wang, Alimohammad Beigi, Bohan Jiang, Amrita Bhattacharjee, Mansooreh Karami, Jundong Li, Lu Cheng, and Huan Liu. 2024. Large language models for data annotation and synthesis: A survey. *arXiv preprint arXiv:2402.13446*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Alpaca: a strong, replicable instruction-following model; 2023. *URL https://crfm. stanford. edu/2023/03/13/alpaca. html*.

Shijie Wang, Wenqi Fan, Yue Feng, Shanru Lin, Xinyu Ma, Shuaiqiang Wang, and Dawei Yin. 2025. Knowledge graph retrieval-augmented generation for llm-based recommendation. *arXiv preprint arXiv:2501.02226*.

Yuming Yang, Yang Nan, Junjie Ye, Shihan Dou, Xiao Wang, Shuo Li, Huijie Lv, Mingqi Wu, Tao Gui, Qi Zhang, and 1 others. 2025. Measuring data diversity for instruction tuning: A systematic analysis and a reliable metric. *arXiv preprint arXiv:2502.17184*.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2025. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*.

Haozhen Zhang, Tao Feng, and Jiaxuan You. 2025a. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. *arXiv preprint arXiv:2506.09033*.

Jieyu Zhang, Ranjay Krishna, Ahmed H Awadallah, and Chi Wang. 2023. Ecoassistant: Using llm assistant more affordably and accurately. *arXiv preprint arXiv:2310.03046*.

Yi-Kai Zhang, De-Chuan Zhan, and Han-Jia Ye. 2025b. Capability instruction tuning: A new paradigm for dynamic llm routing. *arXiv preprint arXiv:2502.17282*.

Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. 2024. Embedllm: Learning compact representations of large language models. *arXiv preprint arXiv:2410.02223*.

A Experimental Setting

A.1 Datasets

Following recent work (Feng et al., 2025), we select data from four distinct tasks, with their statis-

tics summarized in Table 1. The details of these four datasets are outlined as follows:

- **Alpaca** (Taori et al., 2023): A 52k-sample hybrid question-answering dataset used for fine-tuning the Alpaca model. It is automatically constructed by prompting a language model to generate training instances based on a small set of human-written instructions.
- **GSM8K** (Cobbe et al., 2021): A benchmark for evaluating multi-step mathematical reasoning, consisting of 8.5k linguistically diverse grade-school math word problems.
- **SQuAD** (Rajpurkar et al., 2016): A widely used reading comprehension benchmark comprising over 100k question–answer pairs derived from more than 500 Wikipedia articles.
- **Multi-News** (Fabbri et al., 2019): A multi-document summarization dataset containing 56k article–summary pairs. The articles are sourced from Newser.com, and the summaries are professionally written by editors.

A.2 Baseliens

To assess the effectiveness of our data synthesis strategy in the cold-start setting, as well as the performance of our proposed **TRouter**, we conduct extensive comparisons against two categories of baseline methods under distinct experimental scenarios.

- **Smallest LLM**: Always selects the smallest available LLM, prioritizing cost efficiency over performance.
- **Largest LLM**: Always selects the largest available LLM, prioritizing performance regardless of cost.
- **Adaptive LLM**: Selects among the largest, medium, and smallest LLMs based on the user preference scenario. Specifically, it selects the largest LLM for the Performance First scenario, a medium-sized LLM for the Balanced scenario, and the smallest LLM for the Cost First scenario.
- **Prompt LLM**: Incorporates the query, candidate models, and objectives (e.g., prioritizing effectiveness) directly into the prompt, and feeds it into an external LLM (i.e., `gpt-4.1-2025-04-14`) to select the most suitable LLM from a pool of candidates.

In this setting, TRouter is trained exclusively on synthetic data $\mathcal{D}_{\text{syn}}$, generated to simulate realistic training data of the test scenario.

When training data for the test task $\mathcal{D}_{\text{train}}$ is available (i.e., the in-domain setting), TRouter is evaluated against six strong baselines:

- **RouterDC** (Chen et al., 2024): introduces a query-based routing method trained with dual contrastive learning losses, where an encoder and LLM embeddings are jointly optimized to effectively distinguish between capable models, even when multiple LLMs perform well on a given query.
- **GraphRouter** (Feng et al., 2025): models the LLM selection process as an edge prediction task on a heterogeneous graph of tasks, queries, and LLMs, leveraging a heterogeneous GNN to capture contextual interactions and generalize to new LLMs without retraining.
- **FrugalGPT** (Chen et al., 2023): utilizes a pretrained language model to predict the score of the generation result of all LLMs given a query, and then selects the LLM with the highest score within a given cost. We use DistilBert (Sanh et al., 2019) as the router's backbone model.
- **C2MAB-V** (Dai et al., 2024): uses a bandit-based model for LLM selection, which regards each LLM as an arm that implements an exploration mechanism to find a better solution.
- **MetricRouter** (Mei et al., 2025): integrate multiple multilayer perceptron (MLP) layers with sentence transformer embeddings (specifically, the all-MiniLM-L6-v2 model[3]) to develop regression models that estimate both the computational cost and performance of various large language models (LLMs) in response to a given input query.
- **Oracle**: A theoretical upper-bound baseline that assumes access to the ground-truth performance and cost for each query, and always selects the model yielding the highest utility.

[3] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

In this setting, TRouter is trained on $\mathcal{D}_{\text{train}}$, which is realistic training data of the target test scenario. Additionally, each query in $\mathcal{D}_{\text{train}}$ is annotated with the task type at difficulty level.

To further evaluate the generalization ability of existing LLM routers under distribution shift, we assess RouterDC and MetricRouter in a cross-dataset cold-start setting. Specifically, we train on any three of the datasets listed in Table 2 and evaluate on the remaining one. We report the average performance, cost, and utility across all four datasets.

It is essential to note that RouterDC and GraphRouter are classification-based approaches that directly predict the optimal LLM. In contrast, MetricRouter employs a regression-based strategy to estimate both performance and cost, subsequently maximizing utility. In this context, our TRouter can be viewed as a regularized variant of MetricRouter, where the regularization term introduced in Eq. (6) is designed to enhance generalization, particularly under cold-start conditions.

We exclude OmniRouter (Mei et al., 2025) and Carrot (Somerstep et al., 2025) from our comparison, as both adopt the same regression-based paradigm as MetricRouter, predicting performance and cost, then selecting models via utility maximization. However, OmniRouter is tailored for the AIOS setting, which schedules large models to maximize utility over a batch of queries, diverging from our focus on query-level routing. HybridLLM (Ding et al., 2024) is similarly excluded, as it is designed for two-model scenarios and is not directly applicable to our multi-model framework.

A.3 Implementation Details

LLM Usage. For the candidate LLMs used for routing, we primarily select the Qwen 3 series in our main experiments. In addition, we conduct supplementary experiments on several closed-source models, including the Gemini series and the Doubao-Seed models. For Qwen 3 models, we access them via the DashScope API provided by Alibaba Cloud[4]. For the other models, we utilize third-party API providers to obtain model outputs.

For our task-profle guided data synthesis framework, we primarily utilize the `gpt-4.1-2025-04-14` model to instantiate key components, including the Task Type Generator, Task Type Quality Evaluator, and Question-Answer Pair Generator. To validate the generalizability of our approach, we further conduct experiments using the `gemini-2.5-flash` model as a substitute engine for all components of our synthesis framework.

Cost Analysis for Data Synthesis The pricing for `gpt-4.1-2025-04-14` is $2 per million input tokens and $8 per million output tokens, whereas `gemini-2.5-flash` costs $0.30 and $2.50 per million input and output tokens, respectively. During QA generation, `gpt-4.1-2025-04-14` processed approximately 0.91M input tokens and generated 1.56M output tokens, resulting in a cost of $14.34. In contrast, `gemini-2.5-flash` consumed 0.78M input tokens and produced 3.92M output tokens, with a total cost of $10.03. The combined cost of data synthesis was approximately $24.37.

LLM as a Judge for Performance Estimating. Beyond traditional metrics (e.g., token- or subword-level F1 for summarization), we estimate performance using an LLM-as-a-judge (Gu et al., 2024) given the question, the ground-truth answer, and the answer produced by the candidate LLM. This choice is motivated by two considerations. First, prior work typically assumes that the evaluation protocol of the test scenario is known in advance and then derives router training data by applying that protocol to estimate performance. In realistic deployments, especially in the cold-start setting, this assumption may not hold, and a fixed evaluation protocol may be unavailable or misaligned with downstream needs. Second, traditional surface-form metrics exhibit well-documented limitations. For example, F1 implicitly equates lexical overlap with semantic equivalence, whereas summary quality depends on higher-level properties such as content selection, factual consistency, discourse structure, and readability. Our LLM-as-a-judge prompt is provided in Table 14, and we use the `gpt-4.1-nano-2025-04-14` model as the judging LLM to reduce costs.

Details of Task Profile guided Data Synthesis. To construct a task profile-guided synthetic dataset, we leverage `gpt-4.1-2025-04-14` as an LLM engine to instantiate the three key components, including the task type generator, the task type quality evaluator, and the question-answer pair generator. Additional experiments are conducted using `gemini-2.5-flash` to validate the versatility of our framework.

[4] https://bailian.console.aliyun.com/

Since our evaluation spans a wide range of tasks, we begin by aggregating six representative domains from established LLM benchmark[5], involving mathematics, creative writing, commonsense knowledge, programming, long-context understanding, and reading comprehension. These domains serve as seed categories that are expanded to ten broader domains via using the prompt in Table 17 to derive the LLM engine, reducing manual intervention in domain specification.

For each domain, **Task Type Generator** produces up to ten subcategory nodes, and each subcategory is further expanded into no more than five difficulty levels, forming a three-level task taxonomy. The corresponding prompts for the subcategory level and difficulty level are shown in the Table. 18 and Table. 19.

Task Type Quality Evaluator iteratively refines the candidate child task-type set and terminates when no modifications are warranted across three consecutive iterations, and the current set is adopted. The prompts used for task-set quality assessment and refinement are presented in Table 20.

Question-Answer Pair Generator, we configure it to produce questions in batches of eight, generating a total of 40 QA pairs per task profile. We set the semantic similarity threshold to 0.9 in order to remove literally identical or near-duplicate examples. As shown in Appendix D, our diversity analysis confirms that the generated data spans a broad range of topics, primarily due to our multi-level, task-type-guided generation framework. Thus, simple semantic filtering suffices for our objective. Users seeking further refinement may employ more advanced LLM-based filtering methods, though at a significantly higher computational cost. To estimate performance with minimal annotation cost for router training, we employ `gpt-4.1-nano` as a lightweight annotator.

The resulting synthetic dataset $\mathcal{D}_{\text{sys}}$ includes: (i) for `gpt-4.1-2025-04-14`, 10 domains, 103 subcategory nodes, 447 difficulty nodes, and 17,880 QA pairs; (ii) for `gemini-2.5-flash`, the same number of domains but with 98 subcategory nodes, 380 difficulty nodes and 15200 QA pairs.

Detailed of Task-type Guided Routing Framework. In our proposed task-type aware router framework (TRouter), we use the task types at the difficulty level to instantiate $\mathcal{T}$. We utilize all-MiniLM-L6-v2 as the text encoder and map the query and description embedding to a 256-dimensional latent space. We set τ as 0.07. In the cold-start setting, we sample 30 QA pairs per task type for training and 10 for validation. All models are trained using the Adam optimizer with a learning rate of 1e-4, consistent across main experiments. For each model-metric pair, we utilize a two-layer MLP as the regressor to predict the performance and cost.

[5] https://lmarena.ai/leaderboard/text

B Supplementary Theoretical Analysis

In this section, we provide the proof of Eq. (6). We start from the marginal likelihood of the observed triplet (q, h, m) as follows:

$$\log p_\theta(h \mid q, m) = \log \sum_{t \in \mathcal{T}} p_\theta(h, t \mid q, m). \tag{11}$$

After introduce the variational distribution $q_\phi(t \mid q)$ and we rewrite $\log p_\theta(h \mid q, m)$ to:

$$\begin{aligned}\log p_\theta(h \mid q, m) &= \log \sum_t q_\phi(t \mid q)\, \frac{p_\theta(h, t \mid q, m)}{q_\phi(t \mid q)} \\ &= \log \mathbb{E}_{q_\phi(t|q)}\left[\frac{p_\theta(h, t \mid q, m)}{q_\phi(t \mid q)}\right].\end{aligned}$$

By applying Jensen's inequality and because $\log$ is concave, we obtain

$$\log p_\theta(h \mid q, m) \;\geq\; \mathbb{E}_{q_\phi(t|q)}\Big[\log p_\theta(h, t \mid q, m) - \log q_\phi(t \mid q)\Big]. \tag{12}$$

Using the conditional independence assumption $p_\theta(h, t \mid q, m) = p_\theta(h \mid t, m)\, p(t \mid q)$, Eq. (12) becomes

$$\begin{aligned}\log p_\theta(h \mid q, m) \geq\; & \mathbb{E}_{q_\phi(t|q)}[\log p_\theta(h \mid t, m)] \\ & - \mathbb{E}_{q_\phi(t|q)}\left[\log \frac{q_\phi(t \mid q)}{p(t \mid q)}\right].\end{aligned}$$

The second expectation is the Kullback–Leibler divergence. Hence, we further obtain:

$$\boxed{\begin{aligned}\log p_\theta(h \mid q, m) \;\geq\; & \mathbb{E}_{q_\phi(t|q)}[\log p_\theta(h \mid t, m)] \\ & - \mathrm{KL}(q_\phi(t \mid q) \,\|\, p(t \mid q))\end{aligned}} \tag{13}$$

Finally, we prove Eq. (6).

C Supplementary Experiments

C.1 More Ablation Studies

Experiments on closed-source candidate LLMs. Beyond the main experiment where we use the open-source Qwen 3 series as candidate LLMs, we also include supplementary results on closed-source models in Table 8. The results show that

Table 6: Overview of Supplementary Datasets.

Dataset	Task Type	Metric	Cases
WMT (Kocmi et al., 2023)	Translation	BLEU	1000
MBPP (Austin et al., 2021)	Programming	Pass@1	964
LegalBench (Guha et al., 2023)	Legal Domain	F1	1000
MedMcQA (Pal et al., 2022)	Medical QA	F1	1000

Table 7: Ablation study of different designs of TRouter.

Method	Cost	Balance	Performance	Utility Sum
Cold Start				
AgenticRouter	0.0393	0.1771	0.3161	0.5325
Ours w/o Assumptions (1)	0.0357	0.1602	0.3200	0.5159
Ours	0.0355	0.1811	0.3108	0.5274
In-domain				
Ours w/o Assumptions (1)	0.0517	0.1964	0.3445	0.5926
Ours	0.0518	0.1974	0.3433	0.5925

TRouter achieves state-of-the-art performance in both cold-start and in-domain settings on this pool. Furthermore, when trained on the synthesized dataset $\mathcal{D}_{\text{syn}}$, TRouter substantially outperforms Adaptive LLM, providing additional evidence that our task-profile–guided data synthesis is effective for cold-start routing.

Ablation on LLMs for data synthesis. As shown in Table 3, instantiating our task-type synthesis framework with different LLMs (e.g., GPT-4.1, Gemini-2.5-Flash) yields consistent gains and both the task-profile–guided data synthesis and the task-type–aware routing (TRouter) framework outperform baselines across settings. These results indicate these frameworks' versatility with respect to the underlying synthesizer model.

Effect of using Manual label in TRouter. TRouter infers task types via a recognition module. In the cold-start setting, we replace these task type predictions with ground-truth task labels at test time, denoted AgenticRouter in Table 7. AgenticRouter yields further gains over TRouter, but requires expensive LLM-based task classification at inference. In contrast, TRouter dose not require this cost, offering a more efficient solution.

Ablation study of Assumptions (1). We relax Assumption (1) in Sec. 3.3.1, which posits performance is independent of the query given task type and model. Without this assumption. Hence, $p(r \mid q, m)$ can be re-formulated as follows:

$$p(r \mid q, m) = \sum_{t \in \mathcal{T}} p(r \mid t, m, q) p(t \mid q). \tag{14}$$

Results in Table 7 show that the original TRouter (with Assumption 1) consistently outperforms the variant without this assumption and is more robust across different user-preference settings, especially under the cold-start setting.

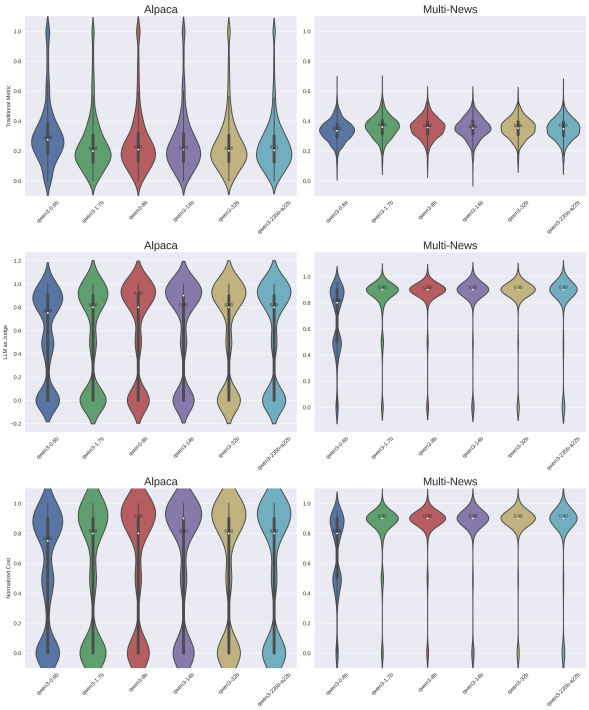

Figure 5: The normalized cost and performance distributions for six Qwen-series models on Alpaca and Multi-News datasets, evaluated using traditional metrics.

Additionally, when releasing both assumption in Eq. (3), the resulting model is MetricRouter. However, as shown in Table 8, Table 4 and Table 3, MetricRouter performs worse than TRouter. Therefore, we suggest that accurately modeling the relationship between each individual query and its associated cost and performance is non-trivial, given the diversity of user queries and the dynamic nature of LLM outputs, especially for a lightweight router model with only a few hundred million parameters. Overfitting to query-specific information (as in MetricRouter) can lead to spurious correlations. While fine-tuning a large language model to capture such query-specific nuances may be feasible, it would drastically increase both latency and computational cost. As such, we conclude that task-type-guided prediction offers a more robust and efficient solution for routing in both cold-start and in-domain settings.

Effect of learning rate. We vary the learning rate of TRouter from 1e-3 to 5e-5 in the cold-start setting and report the average results over five runs in Fig. 6. The results indicate that the routing utility

Table 8: Results across three user preference settings in both cold-start and in-domain scenarios. For in-domain training, candidate LLM performance is obtained using **traditional metrics**. Bold indicate the best results under each user preference, respectively. We use commercial closed-source models in Table 2.

		Cost Preference			Balance Preference			Performance Preference			Utility Sum
Scenario	Method	Cost	Performance	Utility	Cost	Performance	Utility	Cost	Performance	Utility	
Cold-start	Smallest LLM	0.0182	0.3987	0.0652	0.0182	0.3987	0.1903	0.0182	0.3987	0.3153	0.5708
	Largest LLM	0.4473	0.4290	−0.2721	0.4473	0.4290	−0.0092	0.4473	0.4290	0.2537	−0.0275
	Adaptive LLM	0.0182	0.3987	0.0652	0.0157	0.4224	0.2034	0.4473	0.4290	0.2537	0.5223
	Prompt LLM	0.0182	0.3987	0.0652	0.0194	0.4099	0.1953	0.4473	0.4290	0.2537	0.5142
	Ours	0.0165	0.4182	**0.0704**	0.0169	0.4113	**0.1972**	0.0183	0.4044	**0.3199**	**0.5875**
In-domain	RouterDC	0.0154	0.4258	0.0729	0.0186	0.4291	0.2052	0.0169	0.4301	0.3407	0.6188
	GraphRouter	0.0230	0.4197	0.0655	0.0271	0.4086	0.1908	0.0184	0.4174	0.3302	0.5865
	FrugalGPT	0.0259	0.4360	0.0665	0.0509	0.4289	0.1890	0.0528	0.4286	0.3323	0.5878
	C2MAB-V	0.0163	0.4124	0.0694	0.0159	0.4153	0.1997	0.1464	0.4471	0.3284	0.5975
	MetricRouter	0.0137	0.4233	0.0737	0.0167	0.4245	0.2039	0.0231	0.4372	0.3451	0.6227
	Ours	0.0127	0.4311	**0.0760**	0.0140	0.4354	**0.2107**	0.0146	0.4362	**0.3460**	**0.6327**
Oracle		0.0117	0.4891	0.0884	0.0161	0.4975	0.2407	0.0197	0.4994	0.3956	0.7247

Table 9: Results across three user preference settings in both cold-start and in-domain scenarios on four supplementary datasets. For in-domain training, candidate LLM performance is obtained using **traditional metrics**.

		Cost Preference			Balance Preference			Performance Preference			Utility Sum
Scenario	Method	Cost	Performance	Utility	Cost	Performance	Utility	Cost	Performance	Utility	
Cold-start	Smallest LLM	0.0117	0.2977	0.0052	0.0117	0.2977	0.1430	0.0117	0.2977	0.2358	0.4290
	Largest LLM	0.1269	0.3857	−0.0244	0.1269	0.3857	0.1294	0.1269	0.3857	0.2832	0.3882
	Adaptive LLM	0.0117	0.2977	0.0052	0.0384	0.3496	0.1556	0.1269	0.3857	0.2832	0.4890
	Ours	0.0125	0.3211	0.0543	0.0360	0.3490	0.1563	0.0467	0.3642	0.2820	0.4926
In-domain	RouterDC	0.0258	0.3426	0.0479	0.0238	0.3430	0.1596	0.0374	0.3519	0.2740	0.4815
	MetricRouter	0.0147	0.3324	0.0547	0.0563	0.3734	0.1586	0.0457	0.3716	0.2881	0.5014
	Ours	0.0129	0.3242	0.0545	0.0416	0.3757	0.1671	0.0504	0.3766	0.2912	0.5128
	ORACL	0.0151	0.4080	0.0695	0.0199	0.4172	0.1987	0.0289	0.4223	0.3321	0.6002

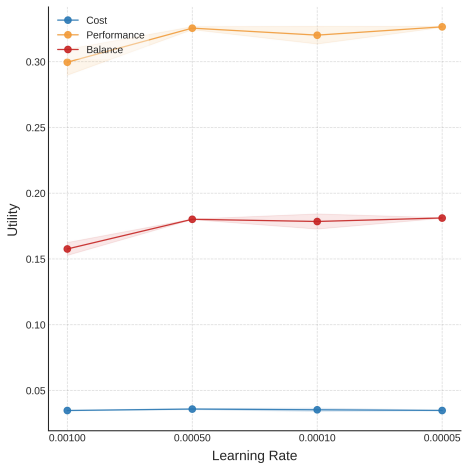

Figure 6: Ablation on learning rate for TRouter in the cold-start setting. For each learning rate setting, results are averaged over five independent runs.

peaks around a learning rate of 1e-4 and remains consistent as the learning rate is further decreased. Finally, we choose 1e-4 as the default learning rate in our main experiments.

Effect of Multi-task-type-label We conduct an additional experiment using `GPT-4.1-2025-04-14` to annotate multi-label task type distributions for each question, in contrast to the single-task-type supervision used in the main setup. The task recognition module is then trained by minimizing the KL divergence between the predicted distribution and the ground-truth multi-label distribution. Specifically, binary label vectors (e.g., [0, 1, 1, 0, ...]) were converted into valid probability distributions using the SoftMax function. Results in Table 11 show no significant performance difference between multi- and single-task-type supervision, likely because our original approach already incorporates probabilistic task assignment through marginalization over all task types via a learned distribution (Eq (4) and Eq (7) in the main paper), effectively capturing task ambiguity and uncertainty.

Experiments on an additional closed-source candidate LLM pool. Beyond the main experiments, we further evaluate TRouter on another closed-source candidate LLM pool to address the concern about model diversity. Specifically, in addition to the model groups used in the main paper, we construct a new pool com-

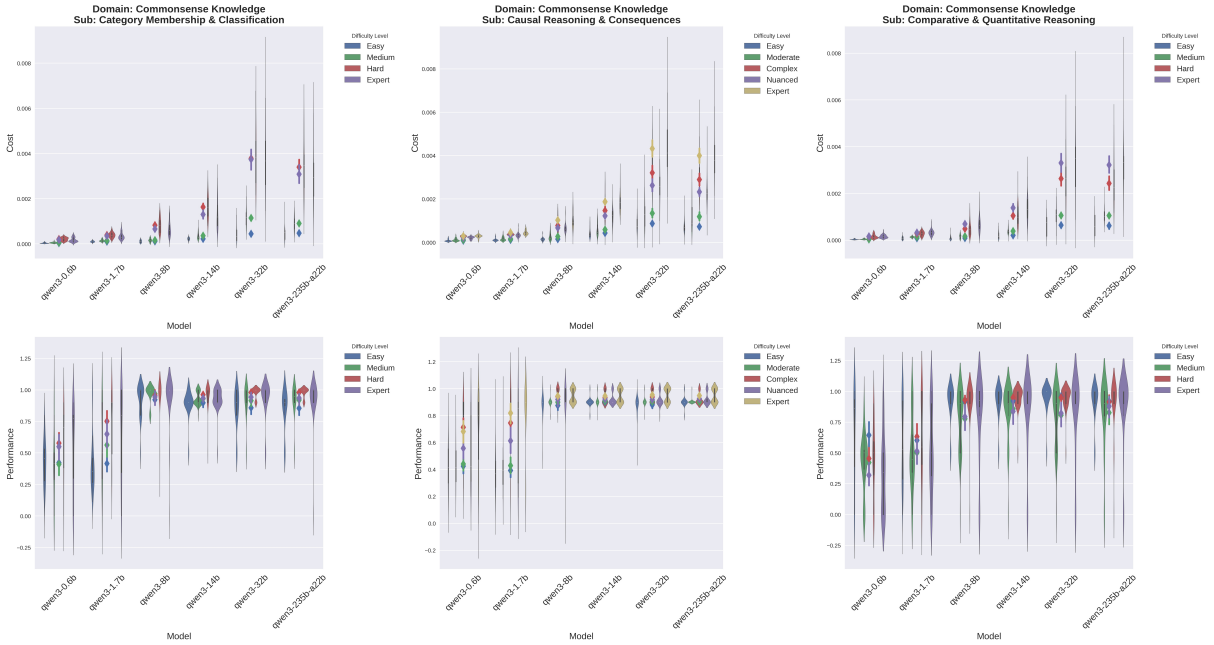

Figure 7: Visualization of cost and performance distributions for six Qwen-series models across representative subcategories of varying difficulty in the Commonsense Knowledge domain, derived from the synthesized dataset $\mathcal{D}_{\text{syn}}$.

Table 10: Results across three user preference settings in both cold-start and in-domain scenarios on an additional closed-source candidate LLM pool. For in-domain training, candidate LLM performance is obtained using **traditional metrics**. Bold indicates the best result for each user preference. The candidate pool consists of `gpt-5`, `gpt-5-mini`, `gpt-5-nano`, and `Doubao-Seed-1.6-Flash`.

		Cost Preference			Balance Preference			Performance Preference			Utility Sum
Scenario	Method	Cost	Performance	Utility	Cost	Performance	Utility	Cost	Performance	Utility	
Cold-start	Smallest LLM	0.0052	0.3987	**0.0756**	0.0052	0.3987	0.1967	0.0052	0.3987	0.3179	0.5902
	Largest LLM	0.4354	0.4573	−0.2689	0.4354	0.4573	0.0109	0.4354	0.4573	0.2788	0.0328
	Adaptive LLM	0.0052	0.3987	**0.0756**	0.0223	0.3992	0.1883	0.4354	0.4573	0.2788	0.5426
	Ours	0.0116	0.4241	0.0755	0.0134	0.4280	**0.2073**	0.0577	0.4350	**0.3365**	**0.6193**
In-domain	RouterDC	0.0099	0.4239	**0.0787**	0.0374	0.4448	0.2030	0.0374	0.4459	0.3492	0.6309
	MetricRoute	0.0100	0.4282	0.0776	0.0135	0.4307	0.2086	0.0586	0.4498	0.3482	0.6344
	Ours	0.0182	0.4451	0.0745	0.0080	0.4328	**0.2124**	0.0280	0.4577	**0.3606**	**0.6474**
Oracle		0.0085	0.4780	0.0888	0.0199	0.4996	0.2399	0.0323	0.5064	0.3987	0.7274

prising `gpt-5`, `gpt-5-mini`, `gpt-5-nano`, and `Doubao-Seed-1.6-Flash`, which span diverse capabilities and cost profiles. We conduct experiments on four datasets from Table 2. The results are reported in Table 10. Similar to the findings in the main paper, TRouter achieves the best overall performance in both cold-start and in-domain settings on this candidate pool. In particular, under cold-start routing, TRouter consistently improves the overall utility compared with Adaptive LLM across different user preferences. In the in-domain setting, TRouter also achieves the highest utility sum, further demonstrating that our framework generalizes well across heterogeneous closed-source model families rather than relying on a specific model series.

Analysis of potential data leakage. To assess the possibility of data leakage, we conduct two analyses on the test splits of GSM8K and SQuAD, using the training split of our synthesized data (30 examples per difficulty level) for comparison. First, we measure literal similarity by computing the cosine similarity between each test example and each synthetic example using embeddings from `all-MiniLM-L6-v2`. For every test sample, we retrieve the top-20 most similar synthetic examples and count cases with similarity equal to 1.0 (exact match) or at least 0.95 (high similarity). We find no exact or high-similarity matches on GSM8K, and only one such case on SQuAD. Second, under

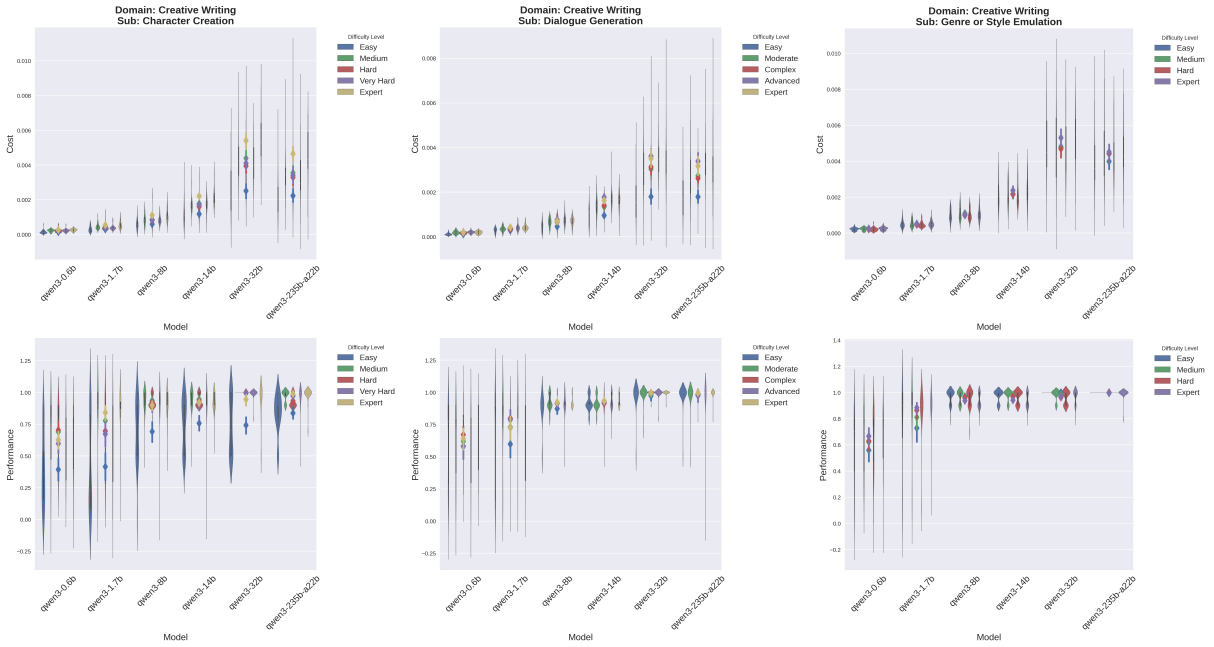

Figure 8: Visualization of cost and performance distributions for six Qwen-series models across representative subcategories of varying difficulty in the Creative Writing domain, derived from the synthesized dataset $\mathcal{D}_{\text{syn}}$.

a stricter criterion, we further examine semantic equivalence, where two questions differ in wording but express the same intent. Specifically, we ask GPT-4.1 to judge whether each benchmark query is semantically equivalent to any of its top-20 retrieved synthetic examples from the first analysis. Under this criterion, no semantically equivalent case is found on GSM8K and only two cases are identified on SQuAD. These results suggest that direct overlap between the synthesized training data and benchmark test sets is extremely limited. Moreover, TRouter already achieves near-optimal routing performance with only 5–10 training examples per difficulty level, substantially below the full 30-shot setting used in Table 4, and it consistently outperforms the baselines in the cold-start setting shown in Tables 3 and 4. Both observations further mitigate the possibility that our results are driven by data leakage. We also note that the similarity-based filtering in our pipeline is only used to remove redundancy within the synthesized data itself, rather than to filter against the test splits.

C.2 Supplementary Dateset

Beyond the four benchmark datasets in the main body, we further evaluate TRouter on MBPP (Austin et al., 2021), MWT'23, MedMCQA (Pal et al., 2022), and LegalBench (Guha et al., 2023), covering four additional domains. We apply the same task-profile–guided data synthesis pipeline and use pass@1 for MBPP, BLEU for MWT, and F1 for MedMCQA and LegalBench. All datasets are split into train/validation/test sets using a 5:1:4 ratio. As shown in Table 9, TRouter again achieves consistently superior performance in both cold-start and in-domain settings, validating the effectiveness of our method.

C.3 Analysis of metric distribution on $\mathcal{D}_{\text{syn}}$.

To examine the cost and performance distribution of models of varying scales on our generated dataset $\mathcal{D}_{\text{syn}}$, we visualize these metrics for six Qwen-series models across representative subcategories of varying difficulty within the Commonsense Knowledge, Creative Writing, and Mathematics domains, derived from the synthesized dataset $\mathcal{D}_{\text{syn}}$ in Figures 7, 8, and 9. The results demonstrate that cost increases monotonically with both model scale and task difficulty, while performance improves with scale but exhibits diminishing returns and declines as difficulty increases, most notably in Mathematics. The 7B and 14B models constitute the price-performance frontier for most subcategories, whereas 32B and larger models provide distinctive value primarily on high-difficulty mathematics and advanced reasoning tasks. These observations support the motivation for our proposed task-type-aware router (TRouter) and task-profile-

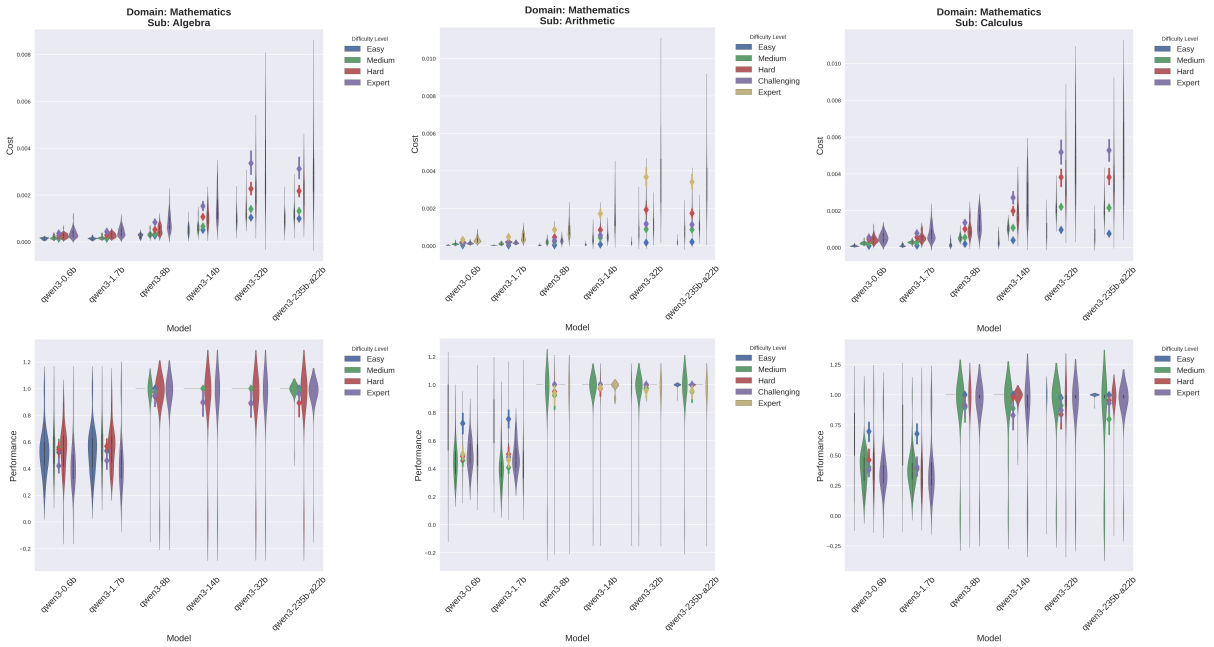

Figure 9: Visualization of cost and performance distributions for six Qwen-series models across representative subcategories of varying difficulty in the Mathematics domain, derived from the synthesized dataset $\mathcal{D}_{\text{syn}}$.

guided data synthesis, which faithfully reproduces complex, real-world cost trends with minimal expense compared to training routers on manually collected in-domain data. Additional analyses are presented below:

Cross-domain patterns Across all domains, cost increases steadily with scale while variance expands for larger models. Mathematics exhibits the steepest high-end cost growth, consistent with longer reasoning chains, while Creative Writing demonstrates the lowest costs at equivalent scales, with Commonsense falling between these extremes. Performance improves with scale across all domains, yet gains diminish beyond 32B parameters. Mathematics shows the highest scale sensitivity, with small models performing markedly worse while 14B and 32B models close the performance gap and can exceed larger models on easier subsets. Creative Writing reaches an early performance ceiling at higher difficulty levels, while Commonsense demonstrates steady, moderate improvements.

Same domain, different subcategories In the Commonsense domain, Category Membership & Classification reaches saturation early with modest cost growth and stable improvements. Causal Reasoning & Consequences requires greater computational resources, with clear advantages for larger models on Hard and Expert difficulty levels and extended cost distributions. Comparative & Quantitative Reasoning falls between these patterns, exhibiting difficulty-driven performance drops more pronounced than those observed in classification tasks.

In Creative Writing, Character Creation and Genre or Style Evaluation show minimal performance gaps at Easy and Medium difficulty levels, diverging only at Hard and Expert levels while maintaining relatively low costs. Dialogue Generation demonstrates greater scale sensitivity, benefiting from larger models at higher difficulty levels while incurring moderately higher costs due to extended output lengths.

In Mathematics, Arithmetic benefits most directly from scaling, with 14B+ models maintaining strong median performance on Hard difficulty and exhibiting pronounced long-tailed cost distributions. Algebra displays greater variance and requires medium-to-large models for Hard and Expert difficulties. Calculus presents the steepest difficulty gradient, where even large models experience performance degradation and increased costs at Expert level.

Fixed domain and subcategory analysis When holding both domain and subcategory fixed, cost increases monotonically with model scale, with larger models displaying heavier-tailed distributions reflecting rare but computationally expensive

samples. Performance improves with model scale, but the cost-performance ratio exhibits diminishing returns. The 1.7B models provide limited gains and demonstrate high sensitivity to difficulty variations. The 7B to 14B range represents the price-performance frontier with the largest marginal benefits. Models at 32B parameters and above continue to improve but with smaller gains, except for notable advantages on high-difficulty mathematics and complex reasoning tasks

Same subcategory, varying difficulty Within a given subcategory, cost increases monotonically from Easy to Expert difficulty levels, with the most pronounced increases observed in Mathematics and the most modest in Creative Writing. Performance exhibits a monotonic decline as difficulty increases, with the steepest degradation occurring in Mathematics, where only larger models maintain positive performance margins on Expert-level tasks. Commonsense tasks show moderate performance decline while continuing to benefit from scaling at higher difficulty levels. Creative Writing demonstrates minimal performance gaps through Hard difficulty, with a more pronounced decline appearing only at Expert level.

Cost–performance implications and methodological relevance The analysis reveals that 7B and 14B models deliver optimal unit-cost performance across most subcategories and difficulty levels, while 32B and larger models should be reserved for applications emphasizing high-difficulty Mathematics or advanced causal and quantitative reasoning tasks. These findings provide direct empirical justification for a task-type-aware routing strategy. Such a router would optimally allocate Easy or Medium-difficulty stylistic writing and classification tasks to 1.7B, 7B, or 14B models, escalating to 32B or larger models only when the task profile indicates reasoning-intensive workloads such as Mathematics-Calculus, Commonsense-Causal reasoning, or Dialogue Generation at Hard or Expert difficulty levels. This adaptive routing strategy reduces heavy-tail computational costs while preserving performance in domains where scaling demonstrates clear benefits (e.g., sometime due to the randomness, we need run LLM for serval times for a given query to obtain satisfactory results).

Furthermore, the observed consistency between cost distributions, difficulty levels, and reasoning complexity substantiates the effectiveness of our task-profile-guided data synthesis approach. This methodology reproduces realistic cost escalation patterns and performance frontiers, enabling the development of benchmarking frameworks and routing policies that transfer effectively to complex real-world deployments while maintaining strong interpretability.

In summary, these empirical observations support the theoretical motivation underlying our proposed task-type-aware router (TRouter) and task-profile-guided data synthesis methodology, which faithfully reproduces complex real-world cost trends with minimal computational overhead compared to training routers on manually collected in-domain data.

C.4 Interpretability Analysis

After being trained on D_{train}, TRoute achieved a top-1 task type prediction accuracy of around 30% on the test set, with a task space size of $|\mathcal{T}| = 447$ while the top-3 accuracy reaches 50% and top-5 accuracy reaches 70%. These results indicate that the latent task-type modeling is able to capture meaningful and discriminative task semantics. Moreover, experimental analysis reveals that most prediction errors originated from the Alpaca dataset, where the top-1 accuracy dropped to just 0.03.

Owing to the interpretability of TRouter, we further investigated the failure cases on the Alpaca dataset by visualizing the top-5 predictions alongside the ground truth in Table 13. The results show that a single test query may correspond to multiple task types. For example, the first case could belong to several types within Information Retrieval. However, since our model uses a weighted combination of task types instead of an argmax selection as defined in Eq. (4), this ambiguity could be mitigated, still leading effective routing decision.

A key advantage of TRouter over black-box embedding-based routers is its interpretable taxonomy structure, which directly facilitates scalability and maintenance through 1) Node Offloading. Users/more advanced LLMs can easily inspect the generated taxonomy. If certain sub-categories are irrelevant to their specific deployment environment, they can simply remove those nodes. 2) Bias Correction. For example, users/more advanced LLMs can observe the routing distribution across the taxonomy. If they detect distributional shift (e.g., a specific node consistently routing to an overly expensive model due to wrong performance prediction), they can manually refine this specific node's

cost/performance parameters. 3) Incremental Updates: When new tasks emerge, the hierarchical nature of our method allows for incremental expansion. Users can generate a new branch for the new task and merge it into the existing tree, rather than re-synthesizing the entire taxonomy from scratch. Overall, TRouter has the potential to evolve further in future applications. However, such discussions are beyond the scope of this work, which focuses specifically on the cold-start scenario. We leave these directions for future exploration.

D Quantitative Analysis of the Quality of Generated QA Pairs

In the cold-start setting, we propose a multi-level task-profile-guided data synthesis framework that requires only domain descriptions to guide LLMs in generating synthetic QA datasets. This approach eliminates the need for manual data collection and annotation. Although prior work shows that low-diversity data offers limited benefit (Yang et al., 2025), and poor-quality synthetic data can harm models or lead to collapse (Shumailov et al., 2024), LLM routing is more robust against these limitations of data synthesis because its objective is to capture cost–performance tradeoffs across models and heterogeneous queries, rather than to execute tasks precisely, the strict quality constraints on training data can be moderately relaxed. Given the large volume of generated data (e.g., 17,880 QA pairs from GPT-4.1-2025-04-14) and the difficulty of some questions, especially in domains like mathematics and commonsense reasoning, which are challenging even for annotators with STEM master's or PhD degrees, it is infeasible to conduct a fully manual quantitative analysis of data quality and diversity. Therefore, to assess the quality of the generated QA pairs, we evaluate the answer accuracy and QA diversity through the following experiments.

Accuracy We focus on three domains, involving Mathematics, Creative Writing, and Commonsense Knowledge. Since answer accuracy is typically correlated with task difficulty, we categorize each subcategory node into three difficulty levels, including easy, medium, and hard. If a node contains more than three difficulty levels, we map all non-extreme levels to medium. For each domain, we sample 160 easy, 160×N medium, and 160 hard questions (where N is the number of difficulty node excepting easy and hard nodes). For commonsense knowledge, due to the breadth of topics and inefficiency of manual retrieval, we use Perplexity for retrieval, followed by human verification. The results show answer accuracy is consistently high across difficulty levels, nearly 100% for easy questions, 97.16% for medium, and 94.5% for hard. Additionally, the accuracy at hard level varies by domain, with hard questions in Mathematics achieving 92.5% and those in Commonsense Knowledge reaching 97.5%. These findings suggest that many of the QA pairs may source from the training data of large models and can be effectively retrieved via generation. This may explain the high accuracy across diverse domains and even at difficulty levels.

Diversity We evaluate diversity at both the subcategory (103 nodes) and difficulty (447 nodes) levels, finding no semantically redundant nodes. While some conceptual similarities exist (e.g.,Long-Form Summarization from Long-context Understanding and Main Idea & Summarization from Reading Comprehension domain), they arise from different emphases across domains or contexts. For instance, the former focuses on long-context compression, while the latter targets reading comprehension.We also examine intra-node diversity by analyzing 30 difficulty nodes from the Progr amming and Commonsense Knowledge domains. Since we remove semantically similar QA pairs during generation, no near-duplicate questions remain . Moreover, each difficulty node covers a diverse range of topics. For example, one programming node covers 12 core topics, including Python multiple inheritance, asynchronous testing, and documentation for complex/generic classes. However, if we consider semantically equivalent questions with different phrasing as redundant, the redundancy rate varies significantly by domain. In the programming domain, semantic redundancy is minimal, whereas in commonsense domains it is approximately 10%.

E Comparison between Synthesized and Manually Labeled Dataset

As outlined in Sec. 3.2, our data synthesis pipeline comprises three components, including the Task Type Generator, Task Type Quality Evaluator, and Question-Answer Pair Generator. Despite multiple attempts, three PhD-level researchers could not produce a diverse and high-quality set of task types, even within their areas of expertise (e.g., programming, creative writing). In contrast, LLMs generate

Table 11: Comparison between single-task-type and multi-task-type training for task recognition.

	Cost First			Balance			Performance			Utility Sum
Task-type for training	Cost	Perf	Utility	Cost	Perf	Utility	Cost	Perf	Utility	
Multi-task-type	0.0414	0.4249	0.0519	0.0409	0.4308	0.1950	0.0678	0.4401	0.3385	0.5853
Single-task-type (Ours)	0.0393	0.4165	0.0518	0.0477	0.4424	0.1974	0.0751	0.4479	0.3433	0.5925

Table 12: Comparison between TRouter trained on LLM-generated data and on manually labeled data.

	Cost First			Balance			Performance			Utility Sum
Method	Cost	Perf	Utility	Cost	Perf	Utility	Cost	Perf	Utility	
Ours	0.0345	0.3098	0.0343	0.0640	0.4261	0.1810	0.0684	0.4224	0.3243	0.5397
Manual Labeled	0.0332	0.3115	0.0357	0.0601	0.4298	0.1849	0.0710	0.4201	0.3219	0.5425

a broad and diverse range of task types efficiently. This is likely because annotating task types requires comprehensive knowledge of domain-specific concepts and the ability to distinguish between varying difficulty levels. Accordingly, we adopt task types generated by LLMs.Moreover, manual construction of a large volume of QA pairs is also impractical. Annotators often lack the expertise to cover specialized topics such as external APIs, mocking, multiple inheritance, or mixed async/sync programming in programming domain. Therefore, we use LLMs to generate candidate QA pairs and then manually verify these data. From these, we select five QA pairs per difficulty node, resulting in a curated dataset of 2,235 QA pairs. For cold-start evaluation, we split the data into 6:4 for training and validation.

As shown in Table 12, performance differences between synthetic and manually labeled datasets are minimal. We argue that is this outcome is expected, as training on data drawn from the same distribution and evaluated under consistent protocols (i.e., using LLM-as-a-judge for both training and test sets) yields more reliable results, as evidenced in Table 4.

F Cost Breakdown Analysis

For synthetic data generation, we employ our multi-level task-profile-guided synthesis framework powered by `GPT-4.1-2025-04-14`. As detailed in Appendix A.3, generating 17,880 QA pairs incurs a total cost of approximately $14.34. In contrast, manual annotation is both economically and practically infeasible. Even PhD-level researchers encountered difficulties in producing diverse task taxonomies and lacked expertise in specialized domains (e.g., complex coding scenarios involving external API mocking).

To estimate manual annotation costs, we assume: 15 minutes per hard QA pair, 3 minutes per medium, and 1 minute per easy question. Unlabeled items are classified as medium difficulty. The dataset includes 103×40 easy, 241×40 medium, and 103×40 hard questions. At a labor rate of $30 per 8-hour day, the total estimated cost exceeds $6,000.

In-domain data collection is also impractical in a cold-start setting, as real user queries are unavailable and recruiting test users would incur costs potentially in the tens of thousands of dollars. Consequently, LLM-based automatic data synthesis is a more viable approach for training task routers under cold-start conditions.

Example 1:
query: Create a list of 4 tips to become a better public speaker.
True Task Type: 'domain': 'Dialogue & Communication', 'subcategory': 'Interactive Tutoring & Socratic Dialogue', 'difficulty': 'Medium'
Definition: Providing brief, step-based guidance or explanation for standard single-step problems with clear solutions.
Top-5 Predictions:
1. Easy (Scientific Reasoning - Scientific Explanation) Possibility: 36.72% Definition: Explaining well-known, fundamental scientific concepts or phenomena using basic language and without requiring prior specialized knowledge.
2. Easy (Reading Comprehension - Fact & Detail Retrieval) Possibility: 19.85% Definition: Identify and extract a single, explicitly stated fact or detail from a sentence, with a direct, unambiguous answer (typically a word or short phrase).
3. Easy (Information Retrieval - Fact Lookup) Possibility: 10.54%
Definition: Retrieval of a single, widely-known fact stated explicitly in authoritative sources.
4. Easy (Information Retrieval - Definition/Explanation Retrieval) Possibility: 3.71%
Definition: Retrieving straightforward definitions or explanations of widely recognized terms or concepts with minimal ambiguity.
5. Moderate (Information Retrieval - Fact Lookup) Possibility: 3.58%
Definition: Retrieval of a less common fact or fact requiring disambiguation (e.g., multiple entities, timeframes, or similar-sounding terms).

Example 2:
query: Choose the correct pairing for the given words.Drawing, music.
True Task Type: 'domain': 'Scientific Reasoning', 'subcategory': 'Concept Comparison', 'difficulty': 'Moderate'
Definition: Comparison of related or similar scientific concepts that may have subtle distinctions or overlapping features, requiring more detailed analysis.
Top-5 Predictions:
1. Easy (Scientific Reasoning - Scientific Explanation) Possibility: 44.46%
Definition:Retrieval of a single, widely-known fact stated explicitly in authoritative sources.
2. Easy (Reading Comprehension - Fact & Detail Retrieval) Possibility: 26.48% Definition: Listing well-known, static items from a single, unambiguous category with no need for filtering or reasoning.
3. Easy (Scientific Reasoning - Scientific Explanation) Possibility: 14.69% Definition: Explaining well-known, fundamental scientific concepts or phenomena using basic language and without requiring prior specialized knowledge.
4. Medium (Information Retrieval - List Generation) Possibility: 4.08%
Definition: Listing items from a category with simple, explicit criteria or filters, requiring only basic fact retrieval and minimal reasoning.
5. Moderate (Information Retrieval - Fact Lookup) Possibility: 2.75%
Definition: Assessing the physical possibility of straightforward, everyday actions or events that rely on well-known physical laws and typical human abilities.

Table 13: Failure Cases of the task recognition module of TRouter.

```
Prompt of LLM-as-Judge
You are an expert evaluator.  Your task is to score the quality and correctness of a
model-generated answer to a given question, using a reference answer as the gold standard.
You will be given:
- QUESTION
- REFERENCE ANSWER (correct)
- MODEL ANSWER (to evaluate)
Your goal is to assign a score between **0.0 and 1.0**, where:
- 1.0 = Fully correct and semantically equivalent to the reference.
- 0.5 = Partially correct or incomplete
- 0.0 = Completely incorrect, irrelevant, or nonsensical
Respond with **only the numeric score**, nothing else.
—
QUESTION:
{question}
REFERENCE ANSWER:
{reference_answer}
MODEL ANSWER:
{response}.
Your score:
```

Table 14: Prompt of LLM-as-Judge. The {question}, {reference_answer}, and {response} will be instantiated by the question, answer, and response from candidate LLMs.

```
TaskTypeGenSystemPrompt = """
You are a knowledge graph construction expert.  Your task is to help build a hierarchical
classification system for an intelligent model router called AgenticiRouter.
AgenticiRouter selects the most appropriate Large Language Model (LLM) based on a user's
query and preferences (e.g., performance, cost-efficiency, latency). To support this, we are
building a multi-level taxonomy of query types.
Each query is classified into a three-level node path:
Domain — broad task area
Subcategory — specific task type
Difficulty Level — complexity level within that task type
"""
```

Table 15: System Prompt of Task type Generation.

```
DomainNodeRule = """
- Domains must be general yet semantically distinct.
- Avoid overlapping or ambiguous categories.
- Think of areas commonly found in LLM benchmarks (e.g., MMLU, AGIEval, CMMLU) and real-world
applications.
- These will serve as the top-level nodes of the taxonomy.
- Only up to {max_new_domain_nodes} new domains can be proposed.
"""

SubcategoryNodeRule = """
- Avoid overlap between subcategories.
- Each subcategory should represent a common type of user query that can be grouped under this
domain.
- Subcategories should be general enough to cover many user queries but specific enough to
guide model selection.
- Only up to {max_subcategory_nodes} new Subcategory Nodes can be proposed for each domain
node.
"""

DifficultyLevelNodeRule = """
- Levels must be ordered from easiest to hardest.
- Levels should reflect increasing reasoning complexity, token usage, or LLM capability
required.
- Levels should be mutually exclusive — no query should belong to more than one level.
- Levels should be collectively exhaustive — all queries in this subcategory must be covered.
- Avoid generic differences like "longer text" unless it reflects actual difficulty in reasoning
or generation.
- Only up to {max_difficulty_level_nodes} Difficulty Level Nodes can be proposed for each
Subcategory Node.
"""
```

Table 16: Rules of nodes generation at three different levels, which will be instantiated to corresponding node generation prompts. {max_new_domain_nodes}, {max_subcategory_nodes}, and {max_difficulty_level_nodes} will be set as 4, 10, 5 in our work.

```
DomainNodePrompt = TaskTypeGenSystemPrompt + """
**Step 1: Generate Domain Nodes**
Generate a list of distinct, non-overlapping **Domain** categories that represent broad types
of user tasks.

We have already defined the following six initial Domains:
- Mathematics
- Creative Writing
- Commonsense Knowledge
- Programming
- Long-context Understanding
- Reading Comprehension

Please propose **{max_new_domain_nodes} new high-level Domains** that:
**Rules:**
{DomainNodeRule}

**Output Format:**
For each proposed Domain:
- Name the Domain
- Provide a one-sentence definition
- Include a real-world example query

**Example:**
<node begin>
Domain: Mathematics
Definition: Covers quantitative problem-solving tasks involving numbers, equations, logic,
or formal systems, including arithmetic, algebra, calculus, and more.
Example: What is the derivative of sin(x)?
Domain: Creative Writing
Definition: Involves imaginative or artistic language generation tasks such as writing poems,
stories, scripts, or creative descriptions.
Example: Write a short story about a robot who learns to paint.
</node end>

**Output:**
"""
```

Table 17: Prompt of domain domain expansion. {max_new_domain_nodes} will be set as 4 in our work. {DomainNodeRule} will be replaced using rules as shown in the Table. 16.

```
SubcategoryNodePrompt = TaskTypeGenSystemPrompt + """
**Step 2: Generate Subcategories for a Given Domain**
The current Domain is: {domain_name}
Domain Definition: {domain_definition}
Domain Example: {domain_example}
Please generate a list of fine-grained Subcategories Nodes that represent specific types of
tasks or problem types within this domain.
Please propose **up to {max_subcategory_nodes} new Subcategory Nodes** for each domain node
that:
**Rules:**
{SubcategoryNodeRule}

**Output Format:**
For each proposed Domain:
- Name the Subcategory
- Provide a one-sentence definition
- Include a real-world example query

Example:
Assume the Domain is:
Domain_name: Mathematics
Domain Definition: Covers quantitative problem-solving tasks involving numbers, equations,
logic, or formal systems, including arithmetic, algebra, calculus, and more.
Domain Example: What is the derivative of sin(x)?
Then the Subcategory Nodes are:
<node begin>
Subcategory: Arithmetic
Definition:  Covers basic arithmetic operations and problem-solving involving numbers,
including addition, subtraction, multiplication, and division.
Example: What is the sum of 123 and 456?
Subcategory: Algebra
Definition:  Involves solving equations and expressions using variables and algebraic
manipulation.
Example: Solve for x in the equation 2x + 3 = 11.
...
</node end>

**Output:**:
"""
```

Table 18: Prompt of subcategory node generation. {max_subcategory_nodes} will be set as 10 in our work. {SubcategoryNodeRule} will be replaced using rules as shown in the Table. 16. {domain_name}, {domain_definition}, and {domain_example} will be replaced by the corresponding domain information.

```
DifficultyNodePrompt = TaskTypeGenSystemPrompt + """
Step 3: Define Difficulty Levels for a Given Subcategory
You are now working on level 3: **Difficulty Level**. This level captures how task complexity
varies **within a specific Subcategory**.

Please utilize a fixed global scale (e.g., Easy / Medium / Hard for three-level difficulty)
as the short name of each level.
However, the Definition of each level should be customized based on the nature of the specific
Subcategory.
**Input Subcategory Information:**
- Domain: {domain_name}
- Subcategory: {subcategory_name}
- Subcategory Definition: {subcategory_definition}
- Subcategory Example Query: {subcategory_example}

Please propose **up to {max_difficulty_level_nodes} Difficulty Level Nodes** that:
**Rules:**
{DifficultyLevelNodeRule}

**Output Format:**
For each proposed level:
- Name the Level
- Provide a one-sentence definition
- Include a real-world example query

**Example**(Subcategory: Code Debugging):
Assume the Subcategory is Code Debugging.
Definition: Identifying and fixing errors in code snippets written in common programming
languages.
Then the Subcategory Nodes are:
<node begin>
Level: Basic
Definition: Single-line or syntax-only bugs in short, self-contained functions.
Example: Fix the indentation error in this 5-line Python function.
Level: Intermediate
Definition: Logic bugs in medium-length code that require control flow analysis.
Example: Fix the loop condition that causes an infinite loop in this JavaScript function.
Level: Advanced
Definition: Bugs across multiple functions, handling of edge cases, or requiring
domain-specific knowledge.
Example: Fix the bug in this Flask app that breaks when uploading empty files.
Level: Expert
Definition: Deep reasoning over large codebases, concurrency, or memory management.
Example: Fix the deadlock issue in this multi-threaded C++ program handling file writes.
</node end>

**Output:**
"""
```

Table 19: Prompt of difficulty node generation. {max_difficulty_level_nodes} will be set as 5 in our work. {DifficultyLevelNodeRule} will be replaced using rules as shown in the Table. 16. {domain_name}, {subcategory_name},{subcategory_definition} and {subcategory_example} will be replaced by the corresponding domain and subcategory information.

```
NodeRevisePrompt = TaskTypeGenSystemPrompt + """
You are given a **candidate {node_name} set** for review. Your job is to:
1. Evaluate whether this candidate {node_name} set needs improvement by checking how well it
adheres to the provided generation rules..
2. If improvement is needed, generate a revised and higher-quality version of the {node_name}
set that better satisfies the rules and supports downstream LLM routing decisions.
Current Candidate {node_name} Set:
{candidate_node_set}
**Node Generation Rules:**
{node_gen_rules}
**Output Format:**
<justification>
Explain whether the current {node_name} set is flawed or could be improved. Mention overlap,
vagueness, gaps, etc.
</justification>
<revision node begin>
{node_name_short}: node name
Definition: One-sentence definition
Example: Example query or task
{node_name_short}: node name
Definition: One-sentence definition
Example: Example query or task
</revision node end>
**Output:**"""

NodeSetChoicePrompt = TaskTypeGenSystemPrompt + """
Your Task: You are given **two candidate {node_name} sets**, labeled **Set A** and **Set B**.
Your job is to:
1. Compare both sets based on how well they follow the generation rules.
2. Select the better set — the one that provides more clarity, distinctiveness, usefulness,
and alignment with routing goals.
3. Justify your choice in detail.
**Node Generation Rules:**
{node_gen_rules}
**Candidate Sets:**
Set A:
{candidate_node_set_a}
Set B:
{candidate_node_set_b}
**Output Format:**
<justification>
Explain why one set is better than the other.  Reference the rules.  Mention clarity,
distinctiveness, coverage, usefulness, etc.
</justification>
<preferred set>
Set A / Set B
</preferred set>
**Output:**
"""
```

Table 20: Prompt of difficulty task type quality evaluator. {node_name} represents the level of task type trees, including domain, subcategory, and difficulty. {node_gen_rules} will be replaced using rules corresponding to {node_name} level, as shown in the Table. 16. {candidate_node_set_a} and {candidate_node_set_b} will be replaced by the original and revised node set.

```
QAGenPrompts = """
You are a helpful assistant that generates high-quality question-answer pairs.
You must respond with a specific format using markers to structure your output.

IMPORTANT: Your response must follow this exact format:

<qa_pairs begin>
Q: What is 2+2?
A: 2+2 equals 4
Q: What is the capital of France?
A: The capital of France is Paris
Q: How does photosynthesis work?
A: Photosynthesis is the process by which plants convert sunlight into energy.
</qa_pairs end>

Rules:
- Start with <qa_pairs begin> and end with </qa_pairs end>
- Each QA pair should be separated by a blank line
- Use "Q:" for questions and "A:" for answers
- Ensure questions are clear, relevant, and the answers are accurate and comprehensive""" +
{TaskProfile} + """Please generate {question_num} different question-answer pairs according
to all the above specifications.
The questions should be clear, relevant, and the answers should be comprehensive and accurate.
Focus on creating diverse questions that cover different aspects of the topic."""
DomainNodeProfile = """
Task Domain: {domain_name}
Domain Definition:
{domain_definition}"""

SubcategoryNodeProfile = """
Task Domain: {domain_name}
Domain Definition: {domain_definition}
Task Subcategory: {subcategory_name}
Subcategory Definition: {subcategory_definition}"""

DifficultyNodeProfile = """
Task Domain: {domain_name}
Domain Definition: {domain_definition}
Task Subcategory: {subcategory_name}
Subcategory Definition: {subcategory_definition}
Task Difficulty: {difficulty_name}
Difficulty Definition: {difficulty_definition}"""
```

Table 21: Prompt of question generator using task profile of different levels. {question_num} will be set as 8 in our work. Other variable with { and } will be replaced with the information of different task types and their parent nodes.