Beyond Query Memorization: Large Language Model Routing with Query Decomposition and Historical Matching

Bo Lv[1,2*], **Jingbo Sun**[2,*], **Jianwei Lv**[2], **Chen Tang**[2], **Shaojie Zhang**[3], **Nayu Liu**[4], **Guoxin Yu**[3], **Zihao Li**[2], **Qichao Zhang**[2], **Dongbin Zhao**[2], **Ping Luo**[2], **Yue Yu**[3 †]
[1]Tencent Hunyuan, [2]University of Chinese Academy of Sciences
[3]Peng Cheng Laboratory, [4]School of Computer Science and Technology, Tianjin University
lvbo19@mails.ucas.ac.cn

Abstract

Optimizing the trade-off among predictive performance and computational cost is a central focus in the deployment of Large Language Models (LLMs). Current routing methods primarily rely on direct mapping from queries to models based on surface-level features, making them susceptible to the memorization trap and leading to poor generalizability on out-of-distribution (OOD) data. In this paper, we propose DecoR, a novel routing framework that recasts the routing task as a matching process of sifting similar queries from historical logs, effectively mitigating the memorization trap. To enhance matching accuracy, we introduce a query capability deconstruction method that decouples linguistic surface forms from task-intrinsic requirements, directing matching toward capability dimensions to ground decisions in essential task attributes. Furthermore, we develop CodaSet, a comprehensive benchmark for assessing routing generalization, where experimental results demonstrate that DecoR maintains superior accuracy while substantially lowering inference costs across both in-distribution and OOD settings. All the codes and data are available at https://github.com/lvbotenbest/DecoR.

1 Introduction

In the practical deployment of Large Language Models (Yang et al., 2025; DeepSeek-AI, 2024), user queries exhibit significant variance in complexity, implying that not all tasks necessitate the involvement of massive-scale models. To this end, Model Routing (Shnitzer et al., 2023; Hu et al., 2024) has gained prominence as a key paradigm, which dynamically selects appropriate model sizes based on query characteristics to optimize the trade-off between predictive performance and computational cost.

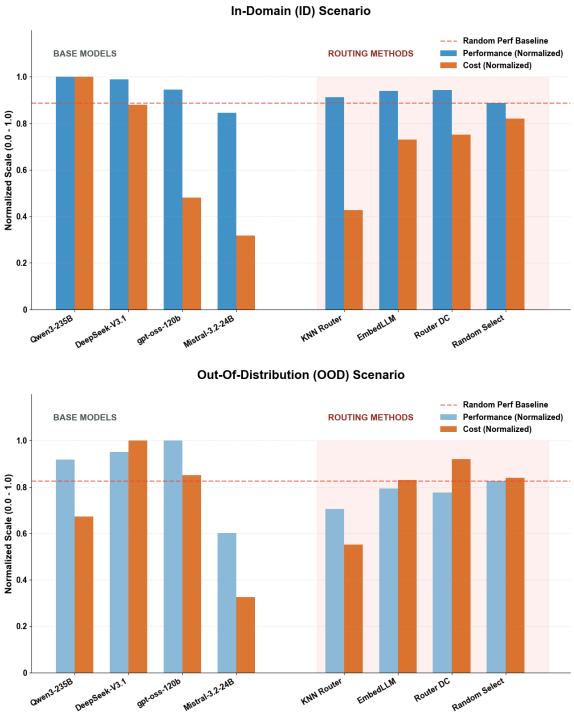

Figure 1: Comparison of routing performance and cost in ID and OOD settings.

However, existing routing methodologies (Chen et al., 2024b; Ding et al., 2024; Zhuang et al., 2025) predominantly simplify the process into a black-box matching task, establishing direct mappings from inputs to model IDs. Such mechanisms are prone to a memorization trap, where routers overrely on surface-level semantics rather than discerning the underlying capability requirements. Consequently, while these methods excel in training-aligned in-domain (ID) scenarios, their generalization collapses on out-of-distribution (OOD) data. As illustrated in Figure 1, when tested on OOD tasks, existing routing methods (Chen et al., 2024b; Hu et al., 2024) fail to surpass the random selection baseline while incurring disproportionately high costs. Furthermore, since these methods (Ong et al., 2024; Chen et al., 2024b; Zhuang et al., 2025)

*Equal contribution.
†Corresponding author.

are typically trained end-to-end for specific model pairs, their decision logic is tightly coupled with the existing model pool. Any update to the underlying models necessitates costly retraining, incurring additional computational and temporal overhead.

To address these challenges, we propose DecoR (**Deco**mposition-based **R**outing), a novel routing framework. Departing from black-box mapping, DecoR decomposes queries into capability requirements to match relevant historical query-response logs, leveraging the model's performance in those matched instances to estimate its prior probability of success for the current query. Specifically, the framework first employs a **Query Deconstruction Stage** to decompose the user query into a structured Capability Profile, encompassing Skills (S), Knowledge (K), and Difficulty (D). Subsequently, the system utilizes the Capability Profile as a core index to identify representative historical query-response logs through a **Hierarchical Sifting Stage** , ensuring these logs are highly aligned with the user query's capability requirements. The underlying rationale is that a model's successful resolution of these matched instances theoretically demonstrates its capability to handle the current query. Finally, the **Empirical Decision Stage** identifies the optimal model by balancing performance and cost within the filtered logs. If sifting yields no matched logs, a fallback safety net deploys a high-performance default model to prevent decision failures in OOD scenarios where prior knowledge is unavailable.

To accurately evaluate routing performance in complex semantic environments and ensure fair comparison, we introduce CodaSet (**C**apability-**O**riented **D**ataset for **A**daptive Routing), a new benchmark constructed using frontier models. Our contributions are as follows:

- We propose DecoR, an innovative routing framework that recasts the routing task into a matching process of sifting similar queries from historical logs, thereby effectively mitigating the memorization trap and enabling system iteration solely through log updates without model retraining.
- We introduce a query capability deconstruction method that decouples linguistic surface forms from task-intrinsic requirements, directing log sifting toward capability dimensions to ground decisions in task attributes and enhance matching accuracy.
- We develop CodaSet, a comprehensive benchmark for assessing the generalization of routing systems. Experimental results show that DecoR consistently maintains superior accuracy while substantially lowering inference costs across both ID and OOD settings.

2 Related Work

The rapid rise of various Large Language Models (LLMs) has spurred the development of model routers (Shnitzer et al., 2023; Hu et al., 2024), which direct simple queries to smaller models to reduce computational costs without compromising overall performance. Lu et al. (2024) proposed a reward-guided routing method that distills reward signals into a routing function to dispatch queries to models with the corresponding expertise. HybridLLM (Ding et al., 2024) utilizes a trained language model as a router to dynamically assign queries to either a small or a large LLM based on predicted task difficulty. Despite its effectiveness, this approach is limited to a binary selection between two models.

To address this constraint, Hu et al. (2024) proposed kNN-Router, a framework that estimates model performance by averaging the outcomes of the k nearest training examples, thereby routing each query to the most suitable LLM from a broader pool. Other recent works have shifted toward representation learning; for instance, Chen et al. (2024b) introduced a dual contrastive learning-based router that jointly optimizes query and model embeddings by aligning queries with compatible models and clustering them semantically. Similarly, Zhuang et al. (2025) developed an encoder-decoder framework to learn compact embeddings for predicting model-query compatibility via a binary cross-entropy objective. Alternatively, Wang et al. (2025) utilized in-context vectors to capture model capabilities, leveraging the relationship between these vectors and query embeddings to predict a model's performance on new queries. In addition, Zhang et al. (2025) explored a text-based approach, transforming candidate model performance into textual descriptions and leveraging a trainable LLM to process these features for dynamic selection.

Despite the success of these methods, most existing approaches rely heavily on learning fixed mappings between query embeddings and model representations. This paradigm risks falling into a memorization trap, where the router tends to memorize

specific training queries rather than generalizing the underlying relationship between task characteristics and model capabilities. To overcome this, we propose DecoR, a framework that recasts the routing task as a matching process of sifting similar queries from historical logs, effectively mitigating the memorization trap. This matching is driven by a capability deconstruction method, which decouples linguistic surface forms from task-intrinsic requirements to significantly enhance accuracy.

3 Method

In this section, we propose DecoR, a model routing architecture designed to derive optimal routing strategies. An overview is illustrated in Figure 2. Following the Problem Formulation (Section 3.1), the Query Deconstruction Stage (Section 3.2) first transforms raw queries into structured capability profiles to capture task-intrinsic requirements. Subsequently, the Hierarchical Log-Sifting Stage (Section 3.3) filters raw historical logs to isolate high-value entries. These are then utilized in the Empirical Decision Stage (Section 3.4) to determine the optimal target model by balancing historical performance and operational cost.

3.1 Problem Formulation

Consider a candidate pool of LLMs $\mathcal{M} = \{m_1, \dots, m_T\}$ and a set of historical response logs $\mathcal{H} = \{(q_i, m_{ij}, r_{ij}) : i = 1, \dots, n\}$, where q_i is a historical query, $m_{ij} \in \mathcal{M}$ is the model invoked, and $r_{ij} = (v_{ij}, c_{ij})$ denotes the corresponding execution result, consisting of the performance score v_{ij} and the operational cost c_{ij}. In practice, multiple models within $\mathcal{M}$ may adequately satisfy the requirements of a specific query, and not every query necessitates the most powerful yet expensive model. Our objective is to learn a router that selects the most suitable LLM $m^* \in \mathcal{M}$ for each incoming query q by identifying a model that offers sufficient performance while minimizing redundant computational cost.

3.2 Query Deconstruction Stage

Traditional routing methods (Ong et al., 2024; Hu et al., 2024) predominantly operate directly on raw queries. However, proximity in semantic space does not necessitate alignment in capability requirements, rendering systems susceptible to routing deviations caused by superficial linguistic features. To address this, we propose **Query Deconstruction**, which aims to decouple **linguistic surface forms** from **task-intrinsic requirements**. Through this deconstruction mechanism, we shift the routing focus from surface-level textual narratives to deep-seated capability demands, ensuring that the decision-making process is grounded in the essential attributes of the task.

Specifically, we develop a Query Deconstructor $f_{dec}(\cdot)$ that transforms each input query q into a structured **Capability Profile** p:

$$p = f_{dec}(q) = \{s, k, d\} \tag{1}$$

This profile quantifies three essential dimensions required to fulfill the query:

- **Skill Set** (S, s_{reason}): $S = \{s_1, s_2, \dots\}$ represents the atomic functional operations (e.g., *information extraction, summarization*) required for q. The accompanying s_{reason} provides a concise explanation justifying why these specific skills are necessary.
- **Knowledge Domain** (K, k_{reason}): $K = \{k_1, k_2, \dots\}$ specifies the domain-specific expertise required (e.g., *medicine, law*). The k_{reason} offers a brief explanation of why these specific knowledge domains are required to address the query.
- **Difficulty** (D, d_{reason}): $D \in \{d_0, d_1, d_2, d_3\}$ quantifies the cognitive load. We discretize this complexity into four hierarchical levels, from trivial requests (d_0) to deep reasoning tasks (d_3). The d_{reason} provides a brief explanation of why the specific difficulty level is assigned to the query.

Notably, the specific categories within the Skill Set and Knowledge Domain are not predefined; instead, they are dynamically derived by the Query Deconstructor based on the unique context of each query. The training process for this deconstructor is elaborated in Section 3.5.

3.3 Hierarchical Log-Sifting Stage

This stage aims to precisely extract the most relevant experiences from a massive experience pool to provide a reliable basis for routing. Initially, the system operates offline to augment the Historical Response Logs with capability dimensions using the Query Deconstructor, forming an enhanced library: $\mathcal{H} = \{(q_i, m_{ij}, r_{ij}, p_i)\}$, where $p_i = \{S_i, K_i, D_i\}$. Upon receiving a new query q_{user} and its capability profile $p_{user} = \{S_u, K_u, D_u\}$,

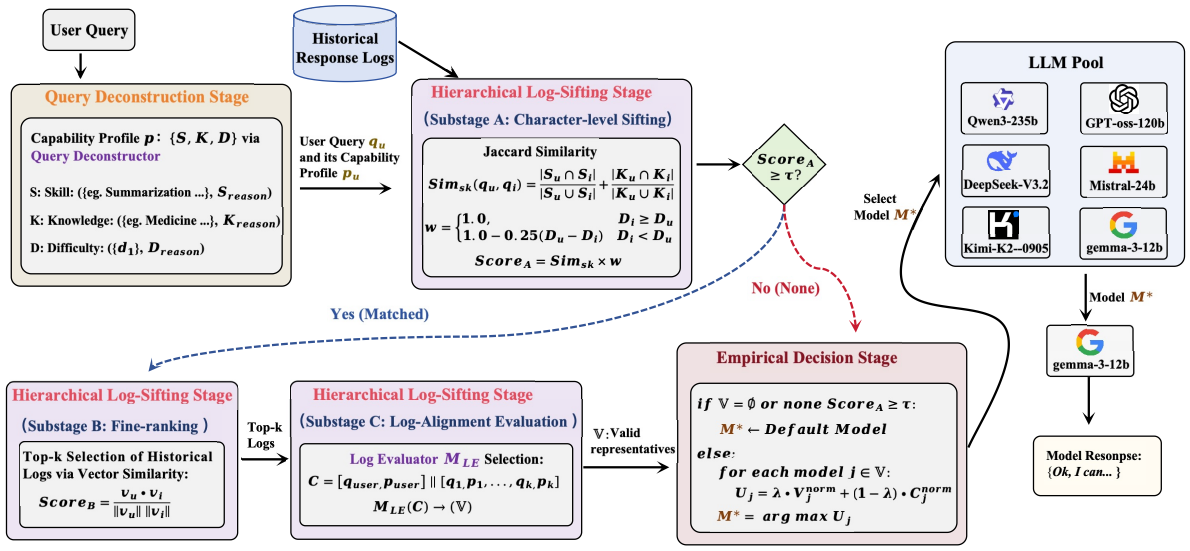

Figure 2: Overview of our DecoR framework. The system deconstructs the input query into a capability profile $p = \{S, K, D\}$. It then identifies representative historical logs with aligned capability requirements through a hierarchical three-tier sifting process. Finally, the framework determines the optimal model M^* by balancing performance and cost.

the system executes the following three progressive sub-stages.

3.3.1 Substage A: Character-level Sifting with Capability Constraints

To ensure retrieval efficiency while enforcing hard capability constraints, the system utilizes an inverted index to match the attributes of q_u against historical logs. We calculate an initial alignment score by independently evaluating the similarity in the skill and knowledge dimensions. Specifically, we employ the Jaccard similarity coefficient to measure the overlap for both Skill Set (S) and Knowledge Domain (K), and define the lexical similarity Sim_{sk} as their summation:

$$Sim_{sk}(q_u, q_i) = \frac{|S_u \cap S_i|}{|S_u \cup S_i|} + \frac{|K_u \cap K_i|}{|K_u \cup K_i|} \quad (2)$$

Subsequently, a difficulty matching function $w(D_i, D_u)$ is introduced to calibrate the initial score. If the difficulty of the historical query is not lower than the current query, it is considered a full match; otherwise, the score decreases by 0.25 for each level of deficiency, such that $w(D_i, D_u)$ is defined as:

$$w = \begin{cases} 1.0, & D_i \geq D_u \\ 1.0 - 0.25 \times (D_u - D_i), & D_i < D_u \end{cases} \quad (3)$$

The final sifting score is defined as $Score_A = Sim_{sk} \times w$. The system employs a preset threshold τ and only retains historical response logs with $Score_A \geq \tau$ for the next stage. If no logs pass this threshold, the user query is identified as out-of-distribution relative to the historical log repository. Consequently, the system bypasses all subsequent sifting stages and proceeds directly to the Empirical Decision Stage (Section 3.4).

3.3.2 Substage B: Fine-ranking

For the candidate logs passing the Substage A (Section 3.3.1), the system performs high-dimensional feature matching to capture deep alignments between semantics and capability features. The BGE-M3 (Chen et al., 2024a) model is used to encode the concatenation of the query text and its capability profile into a feature vector:

$$v = \text{Encoder}(q \oplus \text{String}(p)) \quad (4)$$

The cosine similarity between the target vector v_u and the candidate vector v_i is then calculated:

$$Score_B(v_u, v_i) = \frac{v_u \cdot v_i}{\|v_u\| \|v_i\|} \quad (5)$$

The system ranks the logs in descending order based on $Score_B$ and retains the Top-k most relevant logs for further processing in Substage C (Section 3.3.3).

3.3.3 Substage C: Log-Alignment Evaluation

As the final phase of the sifting process, the **Log Evaluator (LE)** filters the refined logs through long-context modeling to extract a subset of truly

representative records. The input for LE is a concatenated context C of the current requirements and historical experiences:

$$C = [q_{user}, p_{user}] \parallel [q_1, p_1, \ldots, q_k, p_k] \quad (6)$$

The LE model M_{LE} generates a reasoning process (Thought) before outputting the set of identifiers $\mathbb{V}$ representing q_{user}:

$$M_{LE}(C) \rightarrow (\text{Thought}, \mathbb{V}) \quad (7)$$

where $\mathbb{V}$ is the set of indices for logs identified as valid representatives. If the LE determines that no logs in the candidate set provide a valid reference, it outputs $\mathbb{V} = \emptyset$. The training methodology for the Query Deconstructor is elaborated in Section 3.5.

3.4 Empirical Decision Stage

Building on the $\mathbb{V}$, this stage determines the final routing decision. A query is categorized as out-of-distribution (OOD) if it falls into either of the following scenarios: (1) it is pre-identified as OOD during the initial sifting in Substage A (Section 3.3.1), or (2) Substage C (Section 3.3.3) yields an empty set ($\mathbb{V} = \emptyset$). To ensure robustness, the system invokes an fallback strategy for these OOD queries, rerouting them to a high-performance model to guarantee high-quality responses.

If $\mathbb{V} \neq \emptyset$, the decision-maker initiates an empirical inference procedure. To address the magnitude difference between performance scores v_{ij} and inference costs c_{ij}, a normalization procedure is applied to balance their relative influence. Specifically, the system performs empirical data aggregation by retrieving the ground-truth performance of candidate model j from the historical library $\mathcal{H}$ for tasks corresponding to indices in $\mathbb{V}$. The average performance $\bar{V}_j$ and average cost $\bar{C}_j$ for each model are calculated as:

$$\bar{V}_j = \frac{1}{|\mathbb{V}|} \sum_{i \in \mathbb{V}} v_{ij}, \quad \bar{C}_j = \frac{1}{|\mathbb{V}|} \sum_{i \in \mathbb{V}} c_{ij} \quad (8)$$

Subsequently, the system performs dual-dimensional dimensionless processing via linear normalization. This process transforms absolute values into relative scores within the $[0, 1]$ interval. The performance utility score V_j^{norm} and cost utility score C_j^{norm} are defined as:

$$\begin{aligned} V_j^{norm} &= \frac{\bar{V}_j - \min(\bar{V})}{\max(\bar{V}) - \min(\bar{V}) + \epsilon} \\ C_j^{norm} &= \frac{\max(\bar{C}) - \bar{C}_j}{\max(\bar{C}) - \min(\bar{C}) + \epsilon} \end{aligned} \quad (9)$$

where ϵ is an infinitesimal constant to prevent division by zero. This transformation provides a balanced quantitative representation of performance and cost, eliminating the influence of disparate scales.

Finally, a balancing factor $\lambda \in [0, 1]$ is introduced to adjust the preference between performance and economy. The comprehensive utility score U_j for each model is computed as:

$$U_j = \lambda \cdot V_j^{norm} + (1 - \lambda) \cdot C_j^{norm} \quad (10)$$

The routing decision-maker selects the model with the highest utility score as the optimal target:

$$m^* = \arg\max_j U_j \quad (11)$$

where m^* denotes the final model selected to generate the response for the user's query.

3.5 Training

3.5.1 Query Deconstructor

The primary objective of the Query Deconstructor is to decompose a raw query q into a structured capability profile p, as formalized in Eq. (1). Training data are synthesized via GPT-5 and further refined through a rigorous expert review process involving three CS PhD students (detailed protocols and prompts are provided in Appendix A.1). Through this high-quality instruction tuning, the model learns to internalize complex task decomposition patterns. The module is then trained via Supervised Fine-Tuning (SFT) using the following loss function:

$$\mathcal{L}_{dec} = -\sum_{t=1}^{T} \log P(y_t | y_{<t}, q) \quad (12)$$

where q represents the original input query and y denotes the expert-validated structured Capability Profile p.

3.5.2 Log Evaluator

The Log Evaluator implements the mapping defined in Eq. (7) to select a representative log set $\mathbb{V}$ from the input context. To foster autonomous judgment and ensure robust generalization in OOD scenarios, the module is optimized via Group Relative Policy Optimization (GRPO (Shao et al., 2024)). To quantify the alignment between the predicted validation set $\mathbb{V}$ and the ground truth G, we define the reward function $R(\mathbb{V}, G)$ as:

$$\begin{cases} 6, & \text{if } \mathbb{V} = G \\ -2|\mathbb{V}|, & \text{if } G = \emptyset \wedge \mathbb{V} \neq \emptyset \\ -6, & \text{if } G \neq \emptyset \wedge \mathbb{V} \cap G = \emptyset \\ \frac{6}{|G|}(|\mathbb{V} \cap G| - |\mathbb{V} \setminus G|), & \text{otherwise} \end{cases} \quad (13)$$

where $|\mathbb{V} \cap G|$ and $|\mathbb{V} \setminus G|$ denote the number of hits and false positives, respectively. This formulation incentivizes high recall while penalizing hallucinations and retrieval failures. The GRPO training paradigm enables the model to learn the intrinsic utility logic of historical logs rather than relying on rigid classification patterns. The optimization objective is defined as:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{\substack{(q,a)\sim\mathcal{D} \\ o_i \sim \pi_{\text{old}}}} \left[\frac{1}{\sum_{i=1}^{G} |o_i|} \sum_{i=1}^{G} \sum_{t=1}^{|o_i|} L_{i,t} \right] \quad (14)$$

where o_i denotes the i-th sampled output for a given query q; G represents the number of sampled outputs per query, $L_{i,t}$ denotes the loss function:

$$L_{i,t} = \min(r_{i,t}\hat{A}_{i,t},\ \text{clip}\,(r_{i,t},\ 1-\varepsilon,\ 1+\varepsilon)\,\hat{A}_{i,t}), \quad (15)$$

where $r_{i,t}$ is the importance weight and $\hat{A}_{i,t}$ denotes the normalized advantage. In the following, we describe the training data construction process for the Log Evaluator.

Further details regarding the training paradigm and data construction methodology are provided in Appendix A.2.

4 Experiments

4.1 Experiments Setup

4.1.1 Datasets and Metrics

Datasets We construct CodaSet, comprising ID tasks (MMLU-Pro (Wang et al., 2024), GSM8K (Cobbe et al., 2021), IFEval (Zhou et al., 2023), and BBH (Suzgun et al., 2022)) and OOD evaluations (Math500 (Lightman et al., 2023), MT-bench (Zheng et al., 2023), and MBPP (Austin et al., 2021)). Both the model training and our historical log corpus rely exclusively on the ID training sets. During evaluation, the test set encompasses the ID test partitions alongside the complete OOD datasets to assess both task-specific performance and generalization capability. Further details are provided in Appendix B.1.

Evaluation Metrics We employ a diverse set of metrics tailored to each task. For MMLU-Pro, GSM8K and BBH, we use Exact Match (EM) by extracting final answers via regular expressions for ground-truth comparison to calculate accuracy. For specific domains, we leverage established evaluation frameworks: Math500 is assessed using OpenAI's `simple-evals` framework[1], MBPP is evaluated via the EvalPlus[2] to ensure rigorous code correctness. For MT-Bench, we adopt the FastChat LLM-as-a-Judge[3] evaluation protocol and employ GPT-5.1 as the judge to score responses on a 0.1–1 scale. For IFEval, we utilize its official automated script[4] to verify strict constraint adherence. To ensure stability, accuracy-based metrics are determined via majority voting across five independent runs, while MT-bench scores are reported as the arithmetic mean of these iterations.

4.1.2 Implementation Details

We employ **Qwen3-0.6B**[5] as the base model for both the Deconstructor and the Log Evaluator. During the training phase, the Deconstructor is optimized via Supervised Fine-Tuning with a learning rate of 2×10^{-5}. The **Log Evaluator** is trained using the VERL[6] reinforcement learning framework with a learning rate of 1×10^{-6}. Detailed hyperparameter configurations for model training are provided in Appendix B.2. Based on validation set tuning, we set $\tau = 0.5$, $k = 3$, and $\lambda = 0.5$. All results are averaged over three independent trials.

4.1.3 Comparison Methods and LLM Pool

Comparison Methods To ensure a comprehensive evaluation, we compare DecoR with several representative baselines: (1) Random Router (Ong et al., 2024), (2) LLM Router, (3) RouterDC (Chen et al., 2024b), (4) KNN Router (Hu et al., 2024), (5) EmbedLLM (Zhuang et al., 2025), (6) MODEL-SAT (Zhang et al., 2025). All baseline routers are trained using the training split of CodaSet to ensure a fair comparison. Detailed descriptions of these baselines are provided in Appendix B.3.

LLM Pool We construct a diverse LLM pool consisting of eight representative models with varying sizes, including Kimi-K2-Instruct-0905 (Team et al., 2025b), DeepSeek-V3.1-Terminus (DeepSeek-AI, 2024), DeepSeek-V3.2-

[1] https://github.com/openai/simple-evals
[2] https://github.com/evalplus/evalplus
[3] https://github.com/lm-sys/FastChat/tree/main/fastchat/llm_judge
[4] https://github.com/google-research/google-research/tree/master/instruction_following_eval
[5] https://huggingface.co/Qwen/Qwen3-0.6B
[6] https://github.com/volcengine/verl

Exp (DeepSeek-AI et al., 2025), Qwen3-235B-A22B-Instruct (Yang et al., 2025), gpt-oss-120b (OpenAI, 2025), gemma-3-27b-it (Team et al., 2025a), Mistral-Small-3.2-24B-Instruct and gemma-3-12b-it (Team et al., 2025a). All candidate models are accessed through the DeepInfra API[7] to retrieve inference results and collect empirical cost data under real-world deployment settings. Please refer to Appendix B.4 for comprehensive descriptions of these models.

4.2 Main Results

Tables 1 and 3 present the experimental results in both ID and OOD scenarios. In ID settings, our proposed DecoR (DeepSeek-V3.1) significantly outperforms the majority of router baselines while maintaining much lower computational overhead. Specifically, although MODEL-SAT achieves a slightly higher score than DecoR on MMLU-PRO, its computational cost is three times as high. On average, DecoR's performance approximates that of the strongest single model, Qwen3-235B-A22B, and even surpasses it on the IFEVAL benchmark, despite Qwen3's cost being 2.4 times that of our method (5.0x vs. 2.1x). Furthermore, DecoR demonstrates remarkable robustness in OOD scenarios. As shown in Table 3, DecoR experiences only a marginal performance decline in OOD settings, whereas other baseline methods suffer from significant degradation, with some even performing worse than the Random Router. This stability stems from our system's adaptive mechanism: when encountering OOD queries where prior experience is insufficient, the system triggers a fallback logic to a pre-specified high-performance model (DeepSeek-V3.1) rather than making erroneous assignments as the baselines do. Although this strategy leads to a localized increase in cost, it effectively preserves performance stability in unknown domains. In terms of average performance, DecoR remains competitive with top-tier single models while retaining a clear cost advantage.

4.3 Ablation Study

The ablation results in Table 2 demonstrate that the full DecoR system achieves the best performance across all benchmarks. Removing Stage A (Query Deconstruction) causes the most significant decline in accuracy and a sharp cost increase of up to 2.1× on GSM8k, proving that task decomposition is essential for simplifying the reasoning space and reducing redundant computations. While removing Stage B (Fine-ranking) or Stage C (Log-Alignment Evaluation) also leads to noticeable performance drops, their impact on computational overhead is minimal. This confirms the necessity and collective contribution of each module within our proposed framework.

[7] deepinfra.com

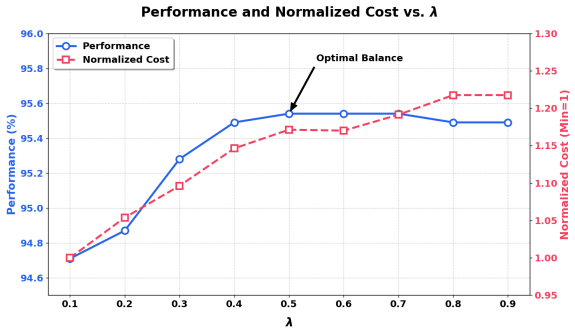

Figure 3: Performance and normalized cost vs. trade-off parameter λ on the GSM8k dataset.

4.4 Analysis

Impact of Base Model Scale The scaling analysis of the Qwen3 backbone reveals that while performance generally improves with model size, the gains are marginal compared to the increased computational overhead. As shown in Table 4, the 4b model achieves only a slight average improvement over the 0.6b variant, specifically 0.11% in ID scenarios and 0.89% in OOD scenarios. Consequently, we select the 0.6b model as our primary backbone to keep the framework lightweight and cost-effective for practical deployment.

Impact of λ on Decision Utility We evaluate the sensitivity of λ, which weights the system's preference for performance over cost on the GSM8k dataset. As illustrated in Figure 3, performance improves substantially as λ increases toward 0.5. At $\lambda = 0.5$, the system achieves an Optimal Balance, maximizing the margin between performance gain and normalized cost. Beyond this threshold ($\lambda > 0.5$), performance plateaus as candidate models reach their inherent capacity; however, the cost escalates sharply as the router increasingly selects expensive models for marginal accuracy gains.

Impact of Shifting Threshold τ in Substage A We explore the sensitivity of Substage A to the threshold τ within the range $[0.1, 0.9]$. As illustrated in Figure 4, increasing τ leads to a gradual improvement in model performance, alongside a corresponding rise in computational cost. In the

Model	GSM8k		MMLU-PRO		BBH		IFEVAL		Average	
	Perf (%)	Cost	Perf (%)	Cost	Perf (%)	Cost	Perf (%)	Cost	Perf (%)	Cost
LLM Pool										
Kimi-K2-Instruct	94.30	11.1x	81.75	11.6x	90.54	13.7x	85.17	12.8x	87.94	11.9x
DeepSeek-V3.1	94.00	5.1x	83.94	5.2x	92.27	5.8x	**88.76**	4.9x	89.74	5.2x
DeepSeek-V3.2-Exp	92.20	4.3x	83.76	4.0x	91.46	4.2x	86.60	2.4x	88.51	3.9x
Qwen3-235B-A22B	**96.40**	3.8x	**84.86**	5.8x	**94.29**	5.4x	86.84	3.3x	**90.60**	5.0x
gpt-oss-120b	94.20	2.4x	80.21	2.8x	92.85	2.7x	87.56	3.1x	88.71	2.7x
gemma-3-27b-it	92.80	1.6x	71.58	3.5x	86.10	1.5x	84.69	1.4x	83.79	2.4x
Mistral-Small-3.2-24B	53.10	2.0x	73.77	1.9x	85.64	1.7x	77.03	**1.0x**	72.39	1.8x
gemma-3-12b-it	94.50	**1.0x**	69.64	**1.0x**	85.35	**1.0x**	81.82	1.2x	82.83	**1.0x**
Router Baselines										
Random Router	87.83	4.1x	77.92	4.8x	88.06	4.7x	82.93	3.0x	84.18	4.4x
LLM Router	94.27	4.3x	65.56	1.1x	84.90	1.1x	84.19	3.6x	82.23	2.3x
KNN Router	93.93	1.2x	77.51	2.5x	92.16	1.6x	82.78	3.3x	86.60	2.0x
RouterDC	90.10	3.8x	79.99	4.4x	91.58	4.4x	85.41	5.0x	86.77	4.2x
EmbedLLM	87.82	4.2x	79.74	4.3x	92.79	4.7x	86.84	3.2x	86.80	4.2x
MODEL-SAT	90.60	5.2x	81.82	6.0x	92.16	5.8x	85.41	3.5x	87.50	7.8x
Proposed Methods										
DecoR (DeepSeek-V3.1)	95.59	1.4x	80.36	2.2x	93.89	1.9x	87.56	4.5x	89.35	2.1x
DecoR (gpt-oss-120b)	95.54	1.3x	79.77	2.0x	93.89	1.9x	87.32	3.0x	89.13	1.9x

Table 1: Performance and cost comparison on the CodaSet ID test set. Performance is quantified as Accuracy (Acc), with raw scores scaled by 100 (%) for clarity. Within the LLM Pool, **bold** values indicate the best performance. For Router Baselines and Proposed Methods, pink represent the highest performance across these two groups, respectively. **Higher performance and lower cost are more desirable.** The notation DecoR (·) signifies the specific model employed as the fallback model.

Method	GSM8k		MMLU-PRO		BBH	
	Perf.	Cost	Perf.	Cost	Perf.	Cost
DecoR (Full)	**95.54**	1.1×	**79.77**	1.0×	**93.89**	1.3×
w/o Satge A	91.70	2.1×	76.16	1.8×	88.57	1.8×
w/o Satge B	94.30	1.0×	77.29	1.1×	90.47	1.0×
w/o Satge C	94.32	1.0×	78.17	1.0×	91.25	1.1×

Table 2: Ablation study of DecoR components on three benchmarks. The best results are **bolded**. Cost is normalized by the column-wise minimum to indicate the relative computational overhead (×).

low τ regime, performance is relatively limited, though costs remain minimal. As τ enters the mid-range, performance gains become significant while cost growth remains moderate. However, in the high τ range, performance tends to saturate while cost increases become more pronounced. Balancing performance enhancement and cost control, we select $\tau = 0.5$ as it achieves an optimal trade-off and serves as our default configuration.

4.5 Case Study

To provide a clearer understanding of the internal decision-making logic of DecoR, we conduct case study experiments. A more comprehensive analysis is provided in Appendix C.1.

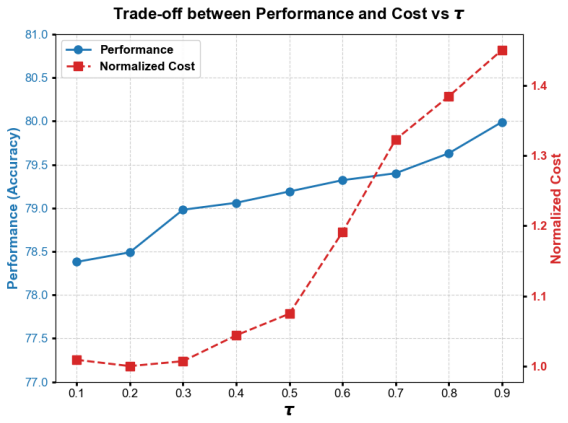

Figure 4: Trade-off between performance and cost across varying τ values. The figure illustrates the trends for model accuracy (solid blue line) and normalized cost (dashed red line) as τ ranges from 0.1 to 0.9. As τ increases, model performance exhibits an upward trend, accompanied by a corresponding increase in computational cost.

5 Conclusion

In this paper, we present DecoR, a routing framework that recasts routing as a log-matching process to effectively mitigate the memorization trap. By decoupling task requirements from surface forms,

Model	Math_500		MBPP		MT_Bench		Average	
	Perf (%)	Cost	Perf (%)	Cost	Perf (%)	Cost	Perf (%)	Cost
LLM Pool								
Kimi-K2-Instruct	81.31	15.2x	77.78	20.1x	94.47	7.9x	84.52	14.9x
DeepSeek-V3.1	89.86	5.6x	75.66	14.1x	97.24	5.3x	87.59	7.1x
DeepSeek-V3.2-Exp	**90.33**	3.5x	75.13	12.2x	95.53	2.4x	87.00	6.8x
Qwen3-235B	85.98	5.3x	**79.37**	13.8x	**98.16**	3.1x	**87.84**	6.4x
gpt-oss-120b	83.18	1.9x	73.54	11.6x	**98.16**	2.9x	84.96	3.8x
gemma-3-27b-it	78.04	1.3x	76.19	1.3x	95.53	1.2x	83.25	1.3x
Mistral-3.2-24B	76.64	1.4x	73.02	6.0x	93.68	1.2x	81.11	2.2x
gemma-3-12b-it	75.70	**1.0x**	76.19	**1.0x**	92.37	**1.0x**	81.42	**1.0x**
Router Baselines								
Random Router	80.18	4.0x	71.96	8.9x	95.26	3.7x	82.47	4.9x
LLM Router	77.57	1.2x	73.19	4.0x	94.34	1.7x	81.70	1.8x
KNN Router	77.10	1.1x	73.54	2.7x	93.95	1.4x	81.53	1.4x
RouterDC	80.64	3.9x	72.66	6.9x	95.00	2.4x	82.77	4.2x
EmbedLLM	82.71	5.1x	73.72	8.4x	94.74	3.2x	83.72	5.4x
MODEL-SAT	81.98	2.9x	74.60	4.7x	94.21	2.0x	83.60	3.3x
Proposed Methods								
DecoR (DeepSeek-V3.1)	85.51	3.1x	76.72	12.4x	97.24	4.7x	86.49	5.0x
DecoR (gpt-oss-120b)	80.37	1.4x	72.49	10.2x	98.16	2.7x	83.67	3.3x

Table 3: Performance and cost comparison of the LLM pool and routing baselines on the CodaSet OOD test set.

Size	GSM8k		MMLU-PRO		BBH		ID Avg	
	Perf.	Cost	Perf.	Cost	Perf.	Cost	Perf.	Cost
0.6b	95.59	1.0×	80.36	1.0×	93.89	1.0×	89.95	1.0×
1.7b	95.59	1.0×	80.42	1.0×	93.89	1.0×	89.97	1.0×
4.0b	**95.63**	1.0×	**80.63**	1.0×	**93.92**	1.0×	**90.06**	1.0×
Size	**Math_500**		**MT_Bench**		**MBPP**		**OOD Avg**	
	Perf.	Cost	Perf.	Cost	Perf.	Cost	Perf.	Cost
0.6b	85.51	1.0×	98.16	1.0×	76.72	1.1×	86.79	1.0×
1.7b	85.51	1.0×	98.16	1.0×	**77.25**	1.0×	86.97	1.0×
4.0b	**85.98**	1.1×	**98.27**	1.1×	77.19	1.1×	**87.15**	1.1×

Table 4: Performance and cost comparison using different sizes of Qwen3 models as the base for Deconstructor and Evaluator. The best performance in each column is **bolded**. Cost is normalized by the column-wise minimum to indicate relative overhead (×).

DecoR grounds decisions in capability dimensions, enhancing both accuracy and robustness. To allow for rigorous evaluation, we introduce the CodaSet benchmark, where DecoR demonstrates superior accuracy and cost-efficiency across both ID and OOD settings. Beyond performance, DecoR enables sustainable system evolution by allowing for seamless iteration via log updates without model retraining.

Limitations

Although DecoR significantly outperforms existing routing baselines in both ID and OOD scenarios while achieving competitive performance at lower costs and offering seamless extensibility through log updates, several areas for optimization remain. First, the scoring process during the construction of historical logs is influenced to some extent by the performance of LLM-as-a-judge. This connection implies that there is still potential for growth in achieving fully automated online updates, where new data could be synchronized into the historical repository in real time during the inference process to continuously enhance the system's knowledge base. We intend to consistently refine this feature as the evaluation capabilities of large language models evolve. Second, the current system architecture could be further improved by incorporating a filtering mechanism for inputs that are highly similar to existing queries in the historical repository to avoid data redundancy. In future research, we will work on developing efficient deduplication and filtering functions to enhance system efficiency and provide more robust support for the research community.

References

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

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024a. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *Preprint*, arXiv:2402.03216.

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

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

DeepSeek-AI. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

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. In *The Twelfth International Conference on Learning Representations*.

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*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

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

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. *Preprint*, arXiv:2406.18665.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *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*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025a. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, and 150 others. 2025b. Kimi k2: Open agentic intelligence. *Preprint*, arXiv:2507.20534.

Chenxu Wang, Hao Li, Yiqun Zhang, Linyao Chen, Jianhao Chen, Ping Jian, Peng Ye, Qiaosheng Zhang, and Shuyue Hu. 2025. Icl-router: In-context learned model representations for llm routing. *Preprint*, arXiv:2510.09719.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, and 1 others. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *Preprint*, arXiv:2311.07911.

Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. 2025. EmbedLLM: Learning compact representations of large language models. In *The Thirteenth International Conference on Learning Representations*.

A Data Synthesis and Expert Review Protocol

A.1 Query Deconstructor

Data Selection and Labeling To construct a high-quality dataset for query decomposition, we leveraged GPT-5.1[8] to synthesize initial query-profile pairs. This process employed a multi-stage prompting strategy as detailed in Table 5. The strategy guided the model to identify core intents, decompose necessary capabilities, and format the output into structured Capability Profiles. This approach ensures that the synthetic data captures the nuanced requirements of complex user queries while maintaining a consistent format for supervised fine-tuning.

Expert Verification Process To ensure the logical integrity of the synthesized samples, we implemented a stringent human-in-the-loop verification process. Three PhD students specializing in Computer Science independently audited each sample based on the criteria of correctness, granularity, and completeness. We adopted a unanimous consensus rule where a sample was incorporated into the final high-fidelity dataset only if it received a "Pass" score from all three experts. This collaborative filtering mechanism effectively eliminated hallucinations and logical inconsistencies, resulting in a reliable training set for the Query Deconstructor.

A.2 Log Evaluator

Data Selection and Labeling To construct a high-fidelity training set for the Log Evaluator, we utilized the filtering results from preceding stages (Stage A and Stage B) to identify logs with varying levels of relevance. We designated the top three historical records as positive samples to represent high-utility evidence. To provide the model with discriminative signals, we randomly selected records ranked significantly lower in the initial retrieval as negative instances. Following the selection, we employed GPT-5.1 to generate the final labels and the underlying reasoning for each sample based on the prompts specified in Table 6. This balanced construction ensures that the model learns to prioritize representative logs that are most conducive to solving the target user query through reinforcement learning.

Expert Verification Process The integrity of the training data was maintained through a rigorous verification process where three PhD students specializing in Computer Science independently audited each sample. Within this protocol, the experts utilized the generated reasoning trajectories as references and incorporated their own professional judgment to assess whether the resulting log sets provided a valid and representative response to the input query. Adhering to a unanimous consensus requirement, a sample was only incorporated into the final dataset if it received an independent pass score from all three experts, while any samples deemed incorrect were discarded. This collaborative audit effectively ensured high data fidelity to provide a reliable foundation for the subsequent GRPO optimization process.

B Experiments Setup

B.1 CodaSet Dataset

B.1.1 Dataset Statistics

The datasets within CodaSet cover a wide spectrum of capabilities, including mathematical reasoning, instruction following, logical deduction, and code generation. For each ID dataset, the data is partitioned into separate training and testing sets. Crucially, to demonstrate that our proposed framework can be seamlessly extended to new domains without the need for additional model training, which effectively allows it to act as a plug-and-play system, we incorporate OOD datasets directly into the evaluation pipeline. For these OOD tasks, the data is used to evaluate the performance of all base models as well as our proposed model. The detailed statistical breakdown of CodaSet is presented in Table 7.

[8] https://platform.openai.com/docs/models/gpt-5.1

	System Prompt:
	You are a Capability Decomposition Engine. Your task is to decompose the user query into its capability-space representation $C(q) = \{S, K, D\}$. Follow all rules strictly and output JSON only.
Template	**Instruction:**
	${query}
	Decomposition Rules:
	1. **Skill Set (S):** Identify required skills (e.g., reasoning, coding). Output as a list and provide "S_reason".
	2. **Knowledge Domain (K):** Identify domains (e.g., law, finance). If none, output "none". Output as a list and provide "K_reason".
	3. **Difficulty (D):** Choose exactly one from {D0, D1, D2, D3} based on complexity. Provide "D_reason".
	Important Rules:
	- Output MUST be valid pure JSON.
	- Do NOT include markdown code.
	Output Format (STRICT):
	{
	"S": [...], "S_reason": "...",
	"K": [...], "K_reason": "...",
	"D": "...", "D_reason": "..."
	}

Table 5: The template used for the Capability Decomposition Engine. The Query Deconstructor also utilizes this exact prompt template for both its training and inference phases

B.1.2 Detailed Dataset Descriptions

- **MMLU-Pro (Wang et al., 2024)**: A more challenging extension of the MMLU benchmark, designed with a larger candidate answer space and harder distractors to better assess models' complex reasoning abilities beyond surface-level knowledge recall.
- **GSM8K (Cobbe et al., 2021)**: A collection of high-quality math word problems that require multi-step reasoning to derive the final answer.
- **IFEval (Zhou et al., 2023)**: A dataset focused on objective verifiable instructions, designed to test the model's ability to strictly adhere to specific formatting constraints and rules.
- **BBH (Big-Bench Hard) (Suzgun et al., 2022)**: A curated subset of the BIG-bench benchmark composed of particularly challenging tasks, designed to evaluate advanced reasoning capabilities beyond surface-level pattern matching.
- **Math_500 (Lightman et al., 2023)**: A benchmark consisting of challenging mathematical problems across multiple subfields, such as algebra and geometry, used to evaluate advanced mathematical reasoning abilities.
- **MT-bench (Zheng et al., 2023)**: A multi-turn conversational benchmark that evaluates dialogue performance across diverse categories using an automated large language model–based judge.
- **MBPP (Austin et al., 2021)**: A benchmark comprising short Python programming problems designed to evaluate programming proficiency, with an emphasis on algorithmic reasoning and code generation.

B.2 Detailed Hyperparameter Configurations

This section provides the comprehensive hyperparameter settings used for training the Query Deconstructor and the Log Evaluator. All experiments in this study were conducted on a server cluster equipped with eight NVIDIA H100 GPUs.

B.2.1 Query Deconstructor Training (SFT)

The Query Deconstructor was fine-tuned using a standard Supervised Fine-Tuning (SFT) pipeline. The detailed parameters are listed in Table 8.

Role	**System Prompt:**
	You are a Query Similarity Judge. Your task is to determine which historical queries can represent the user's query in terms of capability requirements.
Template	**Judgment Criteria:**
	1. **Skills (S):** Historical query skills should cover user query skills (semantic similarity allowed).
	2. **Knowledge (K):** Historical query domains should cover user query domains.
	3. **Difficulty (D):** Historical difficulty must be $\geq$ User difficulty ($D0 < D1 < D2 < D3$).
	User Query & Historical Pool:
	User Query: ${user_query}
	Decomposition: S: ${user_skills}, K: ${user_knowledge}, D: ${user_difficulty}
	Historical Queries:
	${query_a}
	${query_b}
	${query_c}
	Instructions:
	1. Analyze User Query requirements.
	2. Compare each historical query (A, B, C) against the User Query based on S, K, and D.
	3. Determine which queries are valid representatives.
	Output Format (STRICT JSON):
	{
	"thinking": "Brief justification (max 150 words).",
	"valid_representatives": ["A", "B"]
	}

Table 6: The prompt template for the Query Similarity Judge. The Log Evaluator also utilizes this exact prompt template for both its training and inference phases

B.2.2 Log Evaluator Training (RL via verl)

The Log Evaluator was optimized using the `verl` framework. The reinforcement learning hyperparameters are summarized in Table 9.

B.3 Detailed Comparison Methods

We compare **DecoR** against the following baselines:

- **Random Router** (Ong et al., 2024): Selects a candidate LLM uniformly at random for each incoming query.
- **LLM Router**: A prompt-based routing approach that employs an LLM to select models based on natural-language descriptions of their performance characteristics.
- **RouterDC** (Chen et al., 2024b): A dual contrastive learning-based router that jointly trains query and model embeddings by pulling queries toward suitable models while clustering semantically similar queries in the representation space.
- **EmbedLLM** (Zhuang et al., 2025): An encoder–decoder framework that learns compact query and model embeddings to predict model–query compatibility, with the router optimized using a binary cross-entropy objective.
- **MODEL-SAT** (Zhang et al., 2025): A routing method that converts candidate model performance into textual descriptions, which are embedded and processed by a trainable LLM to dynamically select the most suitable model for each query.
- **KNN Router** (Hu et al., 2024): A routing framework that estimates model performance by averaging over the k nearest training examples and routes each query to the LLM with the highest estimated performance.

Category	Dataset	Task Domain	Training Size	Val. Size	Test Size
ID	MMLU-Pro	Multi-discipline	6,756	200	2,734
ID	GSM8K	Math Reasoning	3,200	200	2,000
ID	IFEval	Instruction Following	130	100	300
ID	BBH	Logical Reasoning	4,330	200	1,734
OOD	Math500	Advanced Math	0	0	500
OOD	MT-bench	Multi-turn Chat	0	0	80
OOD	MBPP	Python Coding	0	0	378

Table 7: Statistical distribution and usage of datasets in CodaSet. For ID datasets, validation sets are partitioned from the original training sets.

Hyperparameter	Value
Backbone Model	Qwen3-0.6B
Learning Rate	2×10^{-5}
Batch Size	128
Optimizer	AdamW
LR Scheduler	Cosine
Warmup Ratio	0.03
Weight Decay	0.1
Max Sequence Length	2048
Training Epochs	3
Precision	bf16

Table 8: Hyperparameters for Query Deconstructor SFT.

Hyperparameter	Value
Backbone Model	Qwen3-0.6B
RL Framework	verl
Actor Learning Rate	1×10^{-6}
Train Batch Size	512
Mini Batch Size	256
Rollout Samples per Prompt	8
KL Coefficient	0.001
Max Prompt Length	4096
Max Response Length	2048

Table 9: Hyperparameters for Log Evaluator RL Training.

B.4 Base Models

Detailed profiles of the models in our LLM pool are presented below. For a fair and consistent evaluation, inference cost metrics are derived from DeepInfra's pricing effective as of November 17, 2025, a timeframe that coincides with our data collection and experimental phase.

- **Kimi-K2-Instruct-0905**[9] is a Mixture-of-Experts (MoE) model developed by Moonshot AI, possessing a total of 1 trillion parameters with 32 billion active parameters per forward pass. It is optimized for complex instruction following and large-scale language understanding.

- **DeepSeek-V3.1-Terminus**[10] is a large-scale hybrid reasoning model featuring 671 billion total parameters and 37 billion active parameters. It supports both thinking and non-thinking modes to balance deep reasoning with inference efficiency.

- **DeepSeek-V3.2-Exp**[11] is an experimental iteration toward next-generation architectures featuring 685 billion parameters. It introduces DeepSeek Sparse Attention to validate optimizations for training and inference efficiency in ultra-long context scenarios.

- **Qwen3-235B-A22B-Instruct-2507**[12] is an updated version of the Qwen3 series with 235 billion total and 22 billion active parameters. This version significantly enhances general capabilities in instruction following, logical reasoning, mathematics, and tool usage.

- **gpt-oss-120b**[13] is an open-weight Mixture-of-Experts (MoE) model from OpenAI with

[9] https://deepinfra.com/moonshotai/Kimi-K2-Instruct-0905

[10] https://deepinfra.com/deepseek-ai/DeepSeek-V3.1-Terminus

[11] https://deepinfra.com/deepseek-ai/DeepSeek-V3.2-Exp

[12] https://deepinfra.com/Qwen/Qwen3-235B-A22B-Instruct-2507

[13] https://deepinfra.com/openai/gpt-oss-120b

117 billion parameters. It is designed for high-reasoning tasks, agentic workflows, and general-purpose production use cases.

- **gemma-3-27b-it**[14] is a 27-billion-parameter instruction-tuned model that supports context windows up to 128k tokens. It features improved reasoning and multilingual capabilities across over 140 languages including support for structured outputs.

- **Mistral-Small-3.2-24B-Instruct**[15] is a 24-billion-parameter upgrade over the 3.1 release. It demonstrates markedly better instruction following and a more robust function-calling interface while maintaining high performance across text and vision benchmarks.

- **gemma-3-12b-it**[16] is the 12-billion-parameter variant of the Gemma-3 family. It provides a balanced solution between computational efficiency and advanced reasoning performance for chat-based applications.

C Experimental Analysis

C.1 Case Study

To provide readers with a clearer understanding of the internal decision-making logic of DecoR, we present two representative cases in Table 10 and Table 11 for a comparative analysis:

Case 1: Triggering the Fallback Mechanism (Table 10). This case illustrates the system's rigor when handling domain-specific requirements. The user query involves writing a humorous post about an "Argentinian restaurant." Although the retrieved historical logs overlap partially with the query in terms of formatting requirements, the Log Evaluator accurately identifies the absence of specialized knowledge regarding "Argentinian food culture" and the specific "style imitation" skills required for the target audience. To avoid potential routing errors caused by insufficient prior experience, the system returns an empty set and proactively triggers the fallback mechanism, thereby ensuring performance stability in unfamiliar scenarios.

Case 2: Successful Representative Identification (Table 11). This case demonstrates the system's ability to extract logical commonalities across different contexts. While the user query and the retrieved historical logs involve distinct specific scenarios (such as counting fish versus distributing chocolates), DecoR keenly captures their high consistency in the "arithmetic reasoning" skill dimension and the "D1 difficulty level." This fine-grained, multi-dimensional matching allows the system to effectively reuse historical performance data, enabling an optimal routing decision without the need to blindly invoke high-cost models.

Together, these cases demonstrate how DecoR prevents misjudgments through precise dimensional decomposition and achieves efficient experience reuse when logical cores align, effectively balancing system robustness with cost-efficiency.

[14] https://deepinfra.com/google/gemma-3-27b-it

[15] https://deepinfra.com/mistralai/Mistral-Small-3.2-24B-Instruct-2506

[16] https://deepinfra.com/google/gemma-3-12b-it

Module	Content (Case Study 1: Representative Detection - Empty Output)
User Query	Write a funny post for teenagers about a restaurant called "Buena Onda" which serves Argentinian food. Highlight at least three sections of your response in markdown such as *highlighted section*. Mention "Argentinian" in the post.
Query Decon.	**S:** [writing, creative writing, style imitation, information extraction] **K:** [general knowledge, food culture, Argentinian cuisine] **D:** D1
Historical Logs	**Log A (ID: 14928):** Generate a forum thread about several people waiting to hear the latest local news... *Decon* – **S:** [information extraction, writing, style imitation]; **K:** [general knowledge, online communication]; **D:** D2 **Log B (ID: 14898):** Which of the following is not a fish: salmon or avocado? Answer this easy question first... *Decon* – **S:** [style imitation, writing, information extraction]; **K:** [general knowledge, nutrition, biology]; **D:** D2 **Log C (ID: 14843):** Write a dialogue between two people, one is dressed up in a ball gown and the other is dressed down in sweats... *Decon* – **S:** [creative writing, writing, format imitation, dialogue composition]; **K:** [general knowledge, social interaction]; **D:** D1
Log Evaluator	**Judgment Reasoning:** Query A involves writing and style imitation with a higher difficulty... but lacks creative writing and humor focus. Query B covers style imitation, writing, and information extraction... but its knowledge domain (nutrition, biology) differs significantly. Query C includes creative writing, writing, format imitation... but lacks information extraction and explicit knowledge of food culture or Argentinian cuisine. Since none fully cover the combined skills and knowledge needed for creative, humorous writing about Argentinian food with format elements, none are adequate representatives.
Final Decision	`valid_representatives: []` **Result:** No representative found. The system triggers the **fallback mechanism** to ensure performance stability.

Table 10: Case Study 1: An instance where the DecoR system fails to find a valid representative in the historical logs. Due to the lack of specialized knowledge (Argentinian cuisine) and specific skill combinations in the retrieved candidates, the Log Evaluator returns an empty set, thereby triggering the fallback mechanism to a high-performance base model.

Module	Content (Case Study 2: Representative Detection - Valid Output)
User Query	There's a Bobbit worm hiding in the bottom of James' aquarium. Every day it eats 2 fish. After two weeks, James adds 8 more fish to the aquarium. A week later, he discovers the Bobbit worm. If the aquarium had 60 fish to start with, how many does it have when James discovers the Bobbit worm?
Query Decon.	**S:** [mathematics, logical inference, arithmetic reasoning] **S_reason:** The query requires calculating the number of fish over time using arithmetic and logical reasoning. **K:** [basic mathematics, problem solving] **D:** D1
Historical Logs	**Log A (ID: 2333):** Susan has 3 fish tanks to fill. 1 fish tank contains 7 goldfish and 8 beta fish. The second fish tank contains twice as many fish as the first tank... *Decon* – **S:** [mathematics, logical inference, arithmetic]; **K:** [basic mathematics]; **D:** D1 **Log B (ID: 5454):** Hank gave his wife, Delphine, a box of 24 chocolates... On the first day, Delphine ate 4 chocolates. On the second day, she ate 3 less than twice as many... *Decon* – **S:** [mathematics, arithmetic reasoning, logical inference]; **K:** [basic arithmetic, problem solving]; **D:** D1 **Log C (ID: 5321):** Of the 3 friends, Harry has 4 times as many fish as Joe, and Joe has 8 times as many fish as Sam does. If Sam has 7 fish... *Decon* – **S:** [basic arithmetic, mathematics, logical inference]; **K:** [basic mathematics]; **D:** D1
Log Evaluator	**Judgment Reasoning:** All three historical queries involve arithmetic reasoning, logical inference, and basic mathematics, covering the skills and knowledge required by the user's query. Their difficulty levels are the same as the user's (D1), satisfying the requirement for difficulty. Each historical query involves multi-step calculations with similar or overlapping reasoning and problem-solving concepts, making them suitable representatives of the user's query.
Final Decision	`valid_representatives: ["A", "B", "C"]` **Result:** Successfully matched representatives. The system can leverage the prior performance data of these logs to make an informed routing decision.

Table 11: Case Study 2: An instance where the DecoR system identifies valid representatives. Unlike Case Study 1, the retrieved logs here share identical Skill sets (arithmetic reasoning, logical inference) and Difficulty levels (D1) with the user query. The Log Evaluator confirms that the reasoning patterns are sufficiently similar, allowing the system to reuse historical performance scores instead of triggering the fallback mechanism.