Towards Optimizing SQL Generation via LLM Routing

Mohammadhossein Malekpour **Nour Shaheen** **Foutse Khomh** **Amine Mhedhbi**
Polytechnique Montréal
{mohammadhossein.malekpour,nour.shaheen,foutse.khomh,amine.mhedhbi}
@polymtl.ca

Abstract

Text-to-SQL enables users to interact with databases through natural language, simplifying access to structured data. Although highly capable large language models (LLMs) achieve strong accuracy for complex queries, they incur unnecessary latency and dollar cost for simpler ones. In this paper, we introduce the first LLM routing approach for Text-to-SQL, which dynamically selects the most cost-effective LLM capable of generating accurate SQL for each query.

We present two routing strategies (score- and classification-based) that achieve accuracy comparable to the most capable LLM while reducing costs. We design the routers for ease of training and efficient inference. In our experiments, we highlight a practical and explainable accuracy-cost trade-off on the BIRD dataset.

1 Introduction

In recent years, Text-to-SQL has gained significant momentum with deployments within enterprise solutions to transform data accessibility [1, 13]. Text-to-SQL democratizes access to structured data, allowing non-experts to interact with databases directly without requiring data engineering expertise. For SQL analysts, it enhances their workflows by supporting query authoring, dataset exploration, and report generation. A major use case lies in iterative data exploration, where the *complexity of user queries can range widely—from simple row retrievals to multi-way joins with aggregations*.

Current state-of-the-art Text-to-SQL approaches follow a multi-stage pipeline [5, 7, 9, 16]. The pipeline consists of two main phases: (i) retrieval of contextual information–such as schema elements, examples, and instructions relevant to the query–and (ii) SQL generation. Current enterprise solutions use highly capable LLMs for SQL generation to handle highly complex queries [10, 11]. While this is essential for complex queries, such highly capable LLMs introduce considerable latency and incur higher costs for simpler ones, such as inspecting a few rows from a table. This in turn, can negatively impact both user experience and the average cost per query.

Leading Text-to-SQL benchmarks such as BIRD [8] rank submissions only based on accuracy while also indicating the model used on the leaderboard. Benchmarks typically assume a single-model approach, which mirrors the same inefficiencies found in enterprise deployment—namely, using the most capable models for all queries regardless of complexity. For instance, six of the top ten solutions on BIRD use `GPT-4o` or `Gemini`.

To handle efficiently the varying levels of complexity, *we investigate the implementation of LLM routers for Text-to-SQL*. They route a query to the weakest, yet cheaper and faster model, capable of generating accurate SQL. We propose two routing approaches that achieve an accuracy close to that of the most capable LLM, always outperforming the second-best, and reducing costs by up to $1.4\times$. This cost reduction is substantial for enterprise deployments of analytics or SQL assistants with a high volume of queries. These routers are designed to be easy to train and efficient at inference. Fig. 1 depicts our pipelines with an SQL generation router.

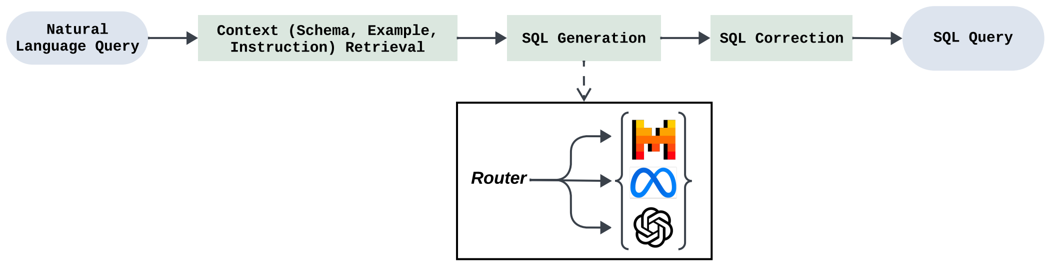


Figure 1: A multi-stage Text-to-SQL pipeline with an LLM router within the generation stage.

2 Preliminaries

2.1 Problem Formulation

Given a set of N models { $\mathcal{M}_0$, $\mathcal{M}_1$, $\ldots$, $\mathcal{M}_{N-1}$ } and a natural language query Q, our objective is to identify the *weakest* model $\mathcal{M}_i$ that can accurately generate corresponding SQL for Q. Here, we define the weakest model as the one with the lowest SQL generation capability. In practice, this model typically exhibits the lowest latency and incurs the least dollar cost. We assume a total ordering of model strength, such that $\mathcal{M}_i$ is considered weaker than $\mathcal{M}_j$ if $i < j$. Therefore, we formulate the problem as an optimization task: select the smallest l such that $\mathcal{M}_l$ produces accurate SQL for Q.

Furthermore, we assume access to a dataset H, consisting of prior natural language (NL) queries, predicted SQL outputs, and corresponding ground-truth SQL for each of the N models. In practice, these examples can be collected from execution logs, user feedback, and manual labeling.

Accordingly, our goal is to learn an N-ary routing function that maps a query Q to the minimal label $l \in [0, N]$. Here, a label $l \in [0, N-1]$ indicates that $\mathcal{M}_l$ is the weakest model capable of generating accurate SQL for Q. A prediction of N denotes that no model within the set is capable of doing so.

2.2 Related Work

LLM Routing has been studied as a binary routing problem [3, 12] for tasks such as question answering, summarization, and information extraction. To our knowledge, this is the first study of LLM N-ary routing and the first for the Text-to-SQL task.

Other prior work uses an ensemble of models for generation, as proposed by Jiang *et al.*, [6]. The approaches use multiple models to generate a set of answers instead of generating a single one and then use pairwise comparison and fusion of top-candidate to generate a final answer. While this approach is promising to maximize accuracy, it goes against the objective of cost minimization. A cascade approach has also been previously employed [2]. However, it cannot be used for Text-to-SQL. Starting with the weakest model, it is infeasible to determine whether the generated SQL is accurate or meets a quality threshold before falling back to the next available stronger model.

2.3 Metrics

To evaluate the effectiveness of generated SQL, we use *execution accuracy* (EX) as the primary metric [8, 15]. Specifically, $EX(S)$ denotes the proportion of queries in the evaluation set S for which the output relation (from the predicted SQL) matches the ground-truth relation, where $0 \leq EX(S) \leq 1$. Here, matching relations are defined as those containing identical tuples, independent of attribute ordering. Our objective is to maximize $EX(S)$ while concurrently choosing the weakest model. In this work, we consider dollar cost as the cost we aim to minimize.

3 Methods

We consider two approaches. The first is a regression approach where we assign a score per model predicting the capability of generating accurate SQL for Q. We say a model is capable of generating accurate SQL if its score is above an input threshold. For all models above that threshold, we pick the weakest. The second is a classification approach where a router model predicts $l \in [0, N]$ as defined in the problem formulation.

If all scores of the regression approach are under the threshold or the classification approach predicts N, then we are predicting that no model is capable of generating accurate SQL. As such, dependent on the use case, we can decide to not attempt to generate SQL or we can route to one of the N models.

3.1 Score-based Routing

A regression-based router predicts for each model $\mathcal{M}_i$, a score $P(EX = 1_{\mathcal{M}_i}|Q)$, *i.e.*, the probability of generating accurate SQL for an input query Q. The router has the input parameter α: the score threshold such that if $P(EX = 1_i|Q) < \alpha$ then $\mathcal{M}_i$ is said to be incapable of generating the query. As such, this router minimizes l such that $P(EX = 1_{\mathcal{M}_l}|Q) \geq \alpha$.

If the scores indicate that no model is capable of generating accurate SQL for Q, we can choose to not make an attempt or to route to one of the models for a different accuracy-cost trade-off. To maximize accuracy in our implementation, we choose to route to the strongest model.

We implement the scoring function using H for each of the N models. Given an input query Q, we first find $\mathcal{Q}_k$: the top-K similar queries in H. For each model $\mathcal{M}$, we set $P(EX = 1_{\mathcal{M}_i}|Q) = EX(\mathcal{Q}_k)_{\mathcal{M}_i}$. We then filter the models $(EX(\mathcal{Q}_k)_{\mathcal{M}_i} \geq \alpha)$ and pick the weakest. We denote this router based on its two parameters as R_α^K.

3.2 Classification-based Routing

We use a distilled `BERT`-style encoder model (`DistilBERT` [14]), which we train on H to learn the function R_{BERT}. Given an input query Q, we input the NL query and relevant retrieved schema to predict the label l indicating the weakest model $\mathcal{M}_l$ predicting $EX(Q) = 1_{\mathcal{M}_i}$. If R_{BERT} predicts N then it is predicting that none of the models can generate accurate SQL. Similarly to the other router, to maximize accuracy in our implementation, we choose to route to the strongest model.

4 Experiments

4.1 Datasets

We conducted our experiments using the BIRD dataset [8], which is widely considered to be the most challenging Text-to-SQL benchmark. Our evaluation set consisted of all 1534 queries in the dev dataset and our training set, acting as H, consisted of 9428 queries.

4.2 Models

Specific models and strength. We used three different LLMs for SQL generation: i) `GPT-4o`; ii) `GPT-4o-mini`; and iii) Llama: `Llama-3.1-8B-instruct-q4_0`. To order their SQL generation capability, we used a simplified Text-to-SQL pipeline consisting of a single attempt at generation while adding the whole schema to the LLM context. We evaluate the pipeline by selecting 10% of the queries of each database in BIRD uniformly at random. We find `Llama` as the weakest model with EX 0.34, then `gpt-4o-mini` with EX 0.48, and `gpt-4o` as the strongest with EX 0.55.

Cost. In our experiments, we only analyzed dollar cost and forgo latency due to setup challenges. We use `Llama` as a local model and hence associate no dollar cost with it and use OpenAI services to access `gpt-4o` and `gpt-4o-mini`. Instead of using current token prices of OpenAI, we use a normalized unit cost where we set `gpt-4o-mini` input and output token cost to 1 and `gpt-4o`to $16.6\times$ (multiplicative price difference).[1] We also used OpenAI's `text-embedding-3-small` model with cosine similarity to find the top K similar queries in H for the score-based router. The embedding cost is negligible per input NL query when compared with token usage and is therefore ignored.

Score-based Router Implementation. For a score-based router (R_k^α), we have to select the two parameters α and K. α (threshold score) is the minimum proportion out of the K similar queries for which a model had to generate accurate SQL to be considerate a candidate model for generation.

[1]Numbers obtained based on OpenAI's offering as of 1 Oct. 2024

Table 1: EX%, model distribution, cost compared to `gpt-4o` for each generation approach on the BIRD dev set, *i.e.*, three routers from above and the use of the three base models: (i) `gpt-4o`; (ii) `gpt-4o-mini`; and (iii) `Llama`: `llama3.1:8b-instruct-q4_0`.

Gen.	**EX%**	**# Queries**				**$ Reduc.**
		`gpt-4o`	`gpt-4o-mini`	`Llama`	`None`	
`4o`	61.02	1534 (100%)	-	-	-	**1x**
`4o-mini`	49.22	-	1534 (100%)	-	-	**16.6x**
`Llama`	29.34	-	-	1534 (100%)	-	∞
$R_{25}^{0.7}$	60.14	197 (13%)	88 (6%)	5 (0%)	1243 (81%)	**1.1x**
$R_{10}^{0.8}$	59.42	160 (10%)	127 (8%)	26 (2%)	1220 (80%)	**1.1x**
$R_{24}^{0.6}$	57.92	324 (21%)	265 (17%)	54 (4%)	890 (58%)	**1.3x**
R_{BERT}	55.21	118 (8%)	311 (20%)	167 (5%)	938 (61%)	**1.4x**

We run a grid search over 10% of train queries as done before and for each input query, we remove the queries in the same database from being chosen as similar ones. We pick $\alpha > 0.5$ as it means intuitively some level of robustness where more than half of the K similar queries were generated successfully. For each $\alpha \in \{0.6, 0.7, 0.8, 0.9\}$, we do a search over $k \in [5, 50]$. We find that setting α to 0.9 is overly restrictive indicating N for every input query, *i.e.*, no model is capable of generate accurate SQL. As such, we only consider $\{0.6, 0.7, 0.8\}$. We choose the K that maximizes EX while having routed to each model at least once. We end up with three models with a difference less than 0.5% EX between them: $R_{24}^{0.6}$, $R_{25}^{0.7}$, and $R_{10}^{0.8}$.

Classification-based Router Implementation. We fine-tuned `DistilBERT` [14] on the fully labeled train set H. We removed from H, all queries in the retail_world database due to schema missing and all duplicate user questions. We split H 80-20 on db_id first (ensuring databases in train are not in validation and vis-versa). This led to a train set of 7346 queries and a validation set of 1697. When training on H and evaluating on the dev set, to retrieved the relevant schema using the TCSL schema linking approach [4, 10]. We trained for 4 epochs, with a batch size of 32, and a learning rate of $1.00E{-}04$. We sampled each batch using a weighted random sampler, the weights correspond to the inverse of the frequency of each label within to the training data.

4.3 Results

We evaluate our routers $R_{24}^{0.6}$, $R_{25}^{0.7}$, $R_{10}^{0.8}$, and R_{BERT} on all 1534 dev queries. Recall that all models, route to the strongest model (`gpt-4o`) in case N is predicted. By analyzing the successful and failed query sets on the three base models (Details in Appendix A), we find that each weaker model has a large subset of failed and successful queries with a stronger one. As such, we expect routing to lower the cost while at best keeping the best model's EX.

Table 1 summarizes the EX, model routing distribution and relative cost to the strongest model (`gpt-4o`) for each generation approach on BIRD's dev set. Our routers are up to 1.4x cheaper while being close in EX. With both routers, we lose some accuracy for a lower cost. The accuracy-cost trade-off on BIRD is easiest to explain through the parameters of the score-based router (Appendix B).

In this ongoing work, we aim next to assess whether an NL query with relevant schema is enough to classify its complexity as done in R_{BERT}. Furthermore, we plan to explore routing across more datasets and within more stages such as correction and schema linking instead of just generation.

5 Conclusion

In this work, we investigated two LLM routing approaches for text-to-SQL on the BIRD benchmark. We believe that cost-based optimization techniques are important for enterprise-level systems where not only accuracy but also cost are critical. In the future, we intend to explore a larger scope of empirical analysis with similar techniques to improve both the execution accuracy and cost.

References

[1] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases - an introduction. *CoRR*, abs/cmp-lg/9503016, 1995.

[2] L. Chen, M. Zaharia, and J. Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *CoRR*, abs/2305.05176, 2023.

[3] D. Ding, A. Mallick, C. Wang, R. Sim, S. Mukherjee, V. Rühle, L. V. S. Lakshmanan, and A. H. Awadallah. Hybrid LLM: cost-efficient and quality-aware query routing. *CoRR*, abs/2404.14618, 2024.

[4] S. T. et al. *CoRR*, abs/2405.16755, 2024.

[5] Z. Hong, Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, and X. Huang. Next-generation database interfaces: A survey of llm-based text-to-sql. *CoRR*, abs/2406.08426, 2024.

[6] D. Jiang, X. Ren, and B. Y. Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *CoRR*, abs/2306.02561, 2023.

[7] B. Li, Y. Luo, C. Chai, G. Li, and N. Tang. The dawn of natural language to sql: Are we fully ready? *CoRR*, abs/2406.01265, 2024.

[8] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Cao, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. C. C. Chang, F. Huang, R. Cheng, and Y. Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *CoRR*, abs/2305.03111, 2023.

[9] X. Liu, S. Shen, B. Li, P. Ma, R. Jiang, Y. Luo, Y. Zhang, J. Fan, G. Li, and N. Tang. A survey of nl2sql with large language models: Where are we, and where are we going? *CoRR*, abs/2408.05109, 2024.

[10] K. Maamari, F. Abubaker, D. Jaroslawicz, and A. Mhedhbi. The death of schema linking? text-to-sql in the age of well-reasoned language models. *CoRR*, abs/2408.07702, 2024.

[11] K. Maamari and A. Mhedhbi. End-to-end text-to-sql generation within an analytics insight engine. *CoRR*, abs/2406.12104, 2024.

[12] I. Ong, A. Almahairi, V. Wu, W. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica. Routellm: Learning to route llms with preference data. *CoRR*, abs/2406.18665, 2024.

[13] A. Quamar, V. Efthymiou, C. Lei, and F. Özcan. Natural language interfaces to data. *Found. Trends Databases*, 11(4), 2022.

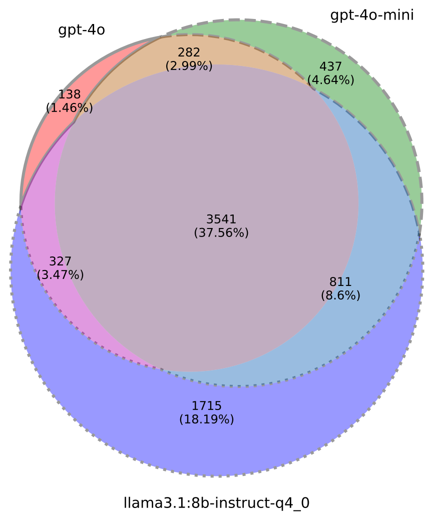
[14] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[15] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *CoRR*, abs/1809.08887, 2018.

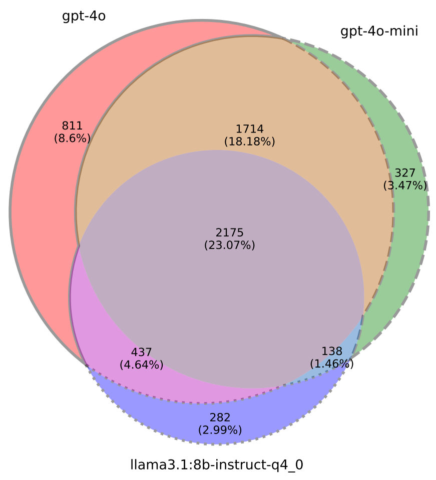
[16] W. Zhang, Y. Wang, Y. Song, V. J. Wei, Y. Tian, Y. Qi, J. H. Chan, R. C.-W. Wong, and H. Yang. Natural language interfaces for tabular data querying and visualization: A survey. *CoRR*, abs/2310.17894, 2024.

A Analysis of Failure Cases

We analyze the performance of each model in terms of correct and failed query sets cases, highlighting their differences in handling SQL generation. Figures 2a and 2b illustrate the distribution of failed and correct predictions across the three models: `gpt-4o`, `gpt-4o-mini`, and `llama3.1:8b-instruct-q4_0`.



(a) Venn Diagram of Failure Cases



(b) Venn Diagram of Correct Cases

Figure 2: Distribution of Failure and Correct Cases across `gpt-4o`, `gpt-4o-mini`, and `llama3.1:8b-instruct-q4_0`

Table 2: Failure Cases across Models

Failure Cases	**Count**	**Percentage**
`gpt-4o`	4288	45.49%
`gpt-4o-mini`	5071	53.79%
`llama3.1:8b-instruct-q4_0`	6394	67.83%
Intersection `gpt-4o` & `gpt-4o-mini` & `llama3.1:8b-instruct-q4_0`	3541	37.56%
Intersection `gpt-4o` & `gpt-4o-mini`	3823	40.55%
Intersection `gpt-4o` & `llama3.1:8b-instruct-q4_0`	3868	41.03%
Intersection `gpt-4o-mini` & `llama3.1:8b-instruct-q4_0`	4352	46.17%

The difference in EX shows a clear gap in capability between models. We find that 37.56% of the queries failed across all three models with a high common failed percentage of queries between every pair from 45.49%–67.83%. This indicates that when routing, it is unlikely to improve EX beyond that of the strongest model and that we expect a cost decrease only.

B Effect of Varying K and α on Execution Accuracy and Model Distribution

We analyze the impact of varying parameters K (number of similar queries) and α (threshold score) in the score-based router R_K^α on execution accuracy (EX) and model distribution. This examination highlights the cost-accuracy trade-off.

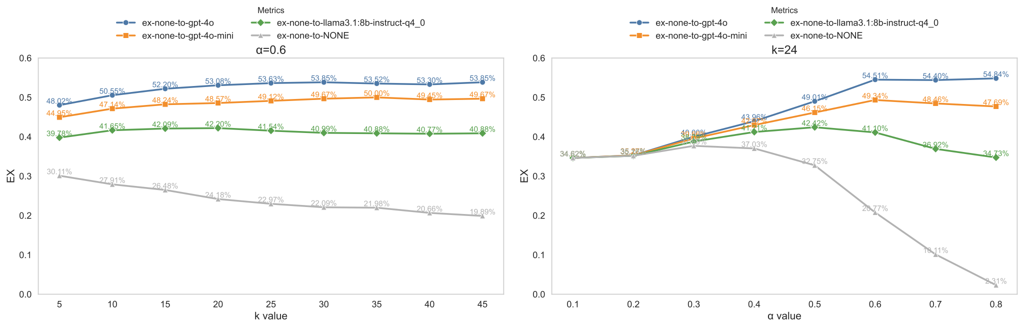

(a) EX as K changes (b) EX as α changes

Figure 3: Execution Accuracy for a score-based router with different None routing strategies.

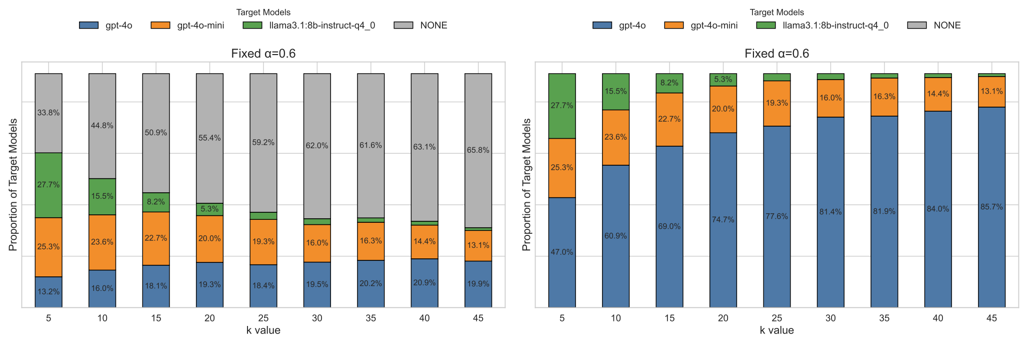

(a) With None prediction (b) None routed to GPT-4o

Figure 4: Effect of varying K on execution accuracy and on model distribution for score-based router.

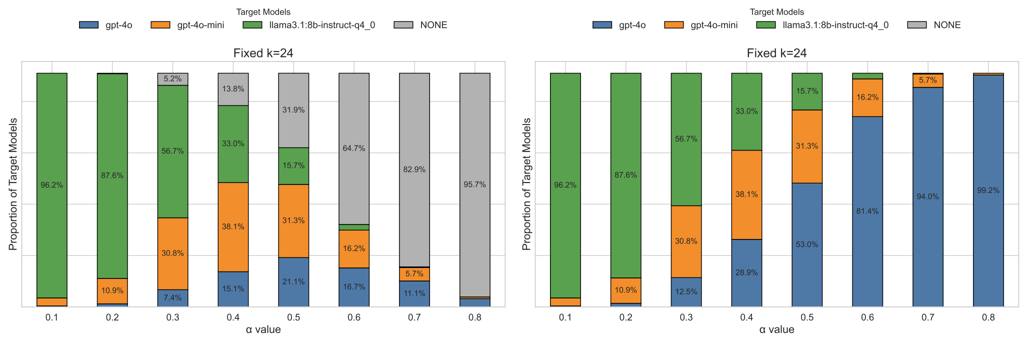

(a) With None prediction (b) None routed to GPT-4o

Figure 5: Effect of varying α on execution accuracy and on model distribution for score-based router.

Our empirical observations are based on 10% of BIRD's train set, summarized in Figures 3, 4, and 5:

- **Higher K or α Values:** leads to an increase execution accuracy (EX) but results in a higher proportion of queries being routed to the strongest mode `gpt-4o`, increasing overall cost.
- **Lower K or α Values:** Favor routing to cheaper models like `gpt-4o-mini`and `Llama`, reducing cost but potentially decreasing EX due to their weaker performance.

In practical applications, selecting appropriate values for K and α is crucial to balance the need for high execution accuracy with cost efficiency. By adjusting these parameters, practitioners can tailor this score-based router to meet their specific requirements.