PORT: Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving

Fangzhou Wu
University of Wisconsin–Madison
fwu89@wisc.edu

Sandeep Silwal
University of Wisconsin–Madison
silwal@cs.wisc.edu

Abstract

Increasing demand for Large Language Models (LLMs) querying services imposes substantial deployment and computation costs. LLM routing offers a cost-efficient solution by directing queries to the optimal LLM based on model and query features. However, existing works focus on offline scenarios and struggle to adapt to online settings with high query volume and constrained token budgets. In this work, we introduce PORT, the first training-free algorithm designed for online routing scenarios. Our algorithm leverages approximate nearest neighbor search to efficiently estimate query features and performs a one-time optimization over a small set of initial queries to learn a routing strategy that guides future routing. We provide theoretical guarantees demonstrating that our algorithm achieves a competitive ratio of $1 - o(1)$ under natural assumptions, which is further validated by extensive experiments across 3 benchmark datasets and 8 baselines, showing an average improvement of **3.55**$\times$ in overall performance, **1.85**$\times$ in cost efficiency, and nearly **4.25**$\times$ in throughput. Our code is available at https://github.com/fzwark/PORT.

1 Introduction

The ability of Large Language Models (LLMs) to effectively interpret diverse domain knowledge has rapidly transformed the landscape of automated information processing [13, 25, 55]. However, the growing volume of user queries imposes substantial deployment costs on LLM-serving providers [10]. For instance, OpenAI reportedly handles up to 12k user queries per second at peak load [40], while its first Azure supercomputer hosted only around 10k GPUs [32], pushing it to rush in more resources to prevent cost spikes [49]. As a result, improving the overall quality of query serving, particularly under limited token budget constraints, has become a critical priority for LLM-serving systems.

One straightforward approach is to route queries with different features to different LLMs while balancing cost and performance [28, 19, 34]. This idea is based on the observation that different LLMs excel in different domains and have varying cost [34], perfectly aligning with the goal of cost-efficient serving. Existing works typically rely on the model-based predictors [34, 41] or computationally complex calculations (e.g., KNN, similarity-weighted ranking) [34, 19] to predict the optimal LLM to route queries to. While effective in offline scenarios, these methods face significant limitations in practical online routing. They are computationally demanding and introduce additional latency, making them impractical for high-volume, low-latency online environments [34, 19]. They do not easily generalize to dynamic LLM deployments, where any configuration change incurs costly retraining overhead [41, 11, 47]. They are also fundamentally limited in achieving effective online routing under constrained token budgets, as they operate under offline assumptions without accounting for the sequential and uncertain nature of real-world query arrivals [35, 38].

Our work instead proposes PORT, the first theoretically grounded online algorithm tailored for high-volume query routing under limited token budget constraints. For each query, we employ Approximate

ML For Systems workshop at Neural Information Processing Systems (NeurIPS 2025).

Nearest Neighbor Search (ANNS) [21, 29] to efficiently estimate its features (performance and cost) for each deployed LLM using a historical dataset. The routing problem (see Appendix A for the formal setup) can be naturally formalized as Mixed-Integer Linear Programming (MILP) that aims to maximize overall performance under token budget constraints. However, solving the global MILP is infeasible in online settings where queries do not arrive simultaneously.

To achieve efficient and near-optimal routing in the realistic sequential setting, we leverage a key observation by looking at the dual problem: At optimality, the dual objective can be fully parameterized by a single dual variable, which directly yields the optimal routing rule. This further motivates us to treat this dual variable as a set of *learnable weights* over LLMs, with the parameterized dual objective serving as the optimization target. Instead of solving the infeasible global dual problem, we adapt it to a partial optimization over a small set of initial observed queries to estimate these weights, which are then applied to guide the routing decisions for subsequent queries. To improve the generalizability of learned weights to future queries, we adopt a random routing strategy for the observed queries and introduce a control parameter into the MILP objective that preserves the optimal solution structure while mitigating performance deviation on unseen queries.

Regarding efficiency, `PORT` only performs a one-time optimization over a small sample set (i.e., observed queries, typically ≈ 250), making it significantly more efficient than existing methods and well-suited for online routing scenarios. In terms of effectiveness, our theoretical analysis guarantees that the proposed algorithm achieves a competitive ratio of $1 - o(1)$ under mild assumptions. This is further supported by extensive experiments, where our method outperforms all 8 baseline methods across diverse routing settings and three benchmark datasets, achieving an average improvement of **3.55**$\times$ in performance, **1.85**$\times$ in cost efficiency, and nearly **4.25**$\times$ in throughput.

To summarize, our main contributions are: **(1)** We propose the **first** training-free online routing algorithm for LLM serving under token budgets. **(2)** It estimates query features via ANNS, ensuring computational **scalability**. **(3)** Operating directly on historical data, it incurs negligible overhead and **scales** across dynamic deployments. **(4)** It requires only a one-time lightweight optimization, demonstrating high **efficiency**. **(5)** We provide theoretical guarantees with a **competitive ratio of** $1 - o(1)$ under mild assumptions. **(6)** Extensive experiments validate these guarantees and show strong **robustness and adaptability**.

Problem Definition. Consider an LLM-serving system with M models, each assigned a token budget B_i (total B). Given a set of incoming queries Q, our goal is to design an online routing strategy $x(\cdot)$ that maximizes response quality under these budget constraints. The offline version of this problem can be naturally formulated as the following MILP (Objective 1), where d_{ij} is the performance score of the response from LLM i to query j, and g_{ij} its token cost. Directly solving Objective 1 in online settings faces key challenges: (i) Unknown ground-truth d_{ij} and g_{ij} without querying LLMs; (ii) Queries arrive sequentially rather than simultaneously in practice. (iii) Computational scalability, high query volume demands routing decisions with low latency, making solving large MILPs or heavy predictions infeasible; (iv) Deployment scalability, varying LLM deployment configurations require adaptive lightweight algorithms.

$$
\begin{aligned}
\max \sum_{j \in Q} \sum_{i \in [M]} & d_{ij} x_{ij} \\
\text{s.t.} \sum_{j} & g_{ij} x_{ij} \leq B_i \ \text{ for all } i, \\
\sum_{i} & x_{ij} \leq 1 \ \text{ for all } j, \\
& x_{ij} \in \{0, 1\}
\end{aligned}
\tag{1}
$$

Motivated by these challenges, we study the following online routing algorithmic problem: Given a larger set of queries Q arriving in a random sequential order, and a predefined token budget constraint, can we design an online routing algorithm that still achieves a near-optimal cumulative performance? Formally, we aim to ensure $\frac{C_{alg}}{C_{opt}} \geq 1 - o(1)$, where we define $C_{alg} := \sum_j \sum_i d_{ij} \hat{x}_{ij}$ denotes the total performance of the algorithm with routing results $\hat{x}_{ij}$, and $C_{opt} := \sum_j \sum_i d_{ij} x^*_{ij}$ denotes the offline MILP optimum with the optimal solution x^*_{ij}.

2 Methodology

2.1 Efficient Performance and Cost Estimation

Our solution to estimate d and g is to leverage a historical dataset $D = \{j, a_j, d_j, g_j\}_{j=1}^n$, where $a_j \in \mathbb{R}^M$ represents the response vector generated by the M LLMs for query j, and $d_j, g_j \in \mathbb{R}^M$

denotes the corresponding performance scores and costs. For any incoming query $j \in Q$, we retrieve nearest neighbors $R_j \subset D$ via ANNS (e.g., HNSW [29]) and compute the estimated performance score $\hat{d}_{ij}$ and cost $\hat{g}_{ij}$ for LLM i as follows: $\hat{d}_{ij} = \frac{1}{|R_j|} \sum_{q \in R_j} d_{iq}$, $\hat{g}_{ij} = \frac{1}{|R_j|} \sum_{q \in R_j} g_{iq}$. One key advantage of widely used ANNS algorithms is their graph-based indexing, which reduces search complexity to about $O(\log |D|)$ in practice, far more efficient than exact KNN ($O(|D|)$) [19].

2.2 Online Routing from Observed Queries

With the approximate features $\hat{d}_{ij}$ and $\hat{g}_{ij}$, we propose an online routing algorithm in Algorithm 1.

Approximate LP with Control Parameter. We first approximate the MILP with estimated features $\hat{d}_{ij}$ and $\hat{g}_{ij}$ and add a control parameter $\alpha > 0$ that only scales the objective in Equation (2). This parameter aids generalization, as discussed in Appendix C.

$$\begin{aligned} \max \sum_{j \in Q} \sum_{i \in [M]} & \alpha \hat{d}_{ij} x_{ij} \\ \text{s.t.} \sum_j & \hat{g}_{ij} x_{ij} \leq B_i, \ \forall i, \\ \sum_i & x_{ij} \leq 1, \ \forall j, \\ & x_{ij} \in \{0,1\}, \ \forall i, j \end{aligned} \tag{2}$$

$$\begin{aligned} \max \sum_{j \in Q} \sum_{i \in [M]} & \alpha \hat{d}_{ij} x_{ij} \\ \text{s.t.} \sum_j & \hat{g}_{ij} x_{ij} \leq B_i, \ \forall i, \\ \sum_i & x_{ij} \leq 1, \ \forall j, \\ & x_{ij} \in [0,1], \ \forall i, j \end{aligned} \tag{3}$$

$$\begin{aligned} \min \sum_{i \in [M]} & \gamma_i B_i + \sum_{j \in Q} \beta_j \\ \text{s.t. } \beta_j & \geq \alpha \hat{d}_{ij} - \hat{g}_{ij} \gamma_i, \ \forall i, j, \\ \gamma_i & \geq 0, \quad \beta_j \geq 0, \ \forall i, j \end{aligned} \tag{4}$$

Dual LP under Relaxation. We relax Equation (2) by allowing x_{ij} to take fractional values in $[0, 1]$. This yields the relaxed approximate LP in Equation (3) with its dual given in Equation (4).

Routing via Learned γ^*. By complementary slackness, if x is the optimal solution to Equation (3) and (γ, β) is optimal solution to its dual, then $x_{ij} > 0 \Leftrightarrow \beta_j = \max_i (\alpha \hat{d}_{ij} - \hat{g}_{ij} \gamma_i)$. This implies that, at optimality, the dual objective can be expressed as a function parameterized by γ: $F(\gamma) = \sum_i \gamma_i B_i + \sum_j \max_i (\alpha \hat{d}_{ij} - \hat{g}_{ij} \gamma_i)$ (5). Ideally, queries should be routed to the i maximizing $\alpha \hat{d}_{ij} - \hat{g}_{ij} \gamma_i$, but computing the global optimal γ offline is infeasible. We therefore adapt this idea to the online setting: we learn an estimated set of weights γ^* from the first $P = \epsilon |Q|$ queries by minimizing $F(\gamma, P) = \epsilon \sum_i \gamma_i B_i + \sum_{j \in P} \max_i (\alpha \hat{d}_{ij} - \gamma_i \hat{g}_{ij})$ (6). For each $j \in P$, we compute $\hat{d}_{ij}$ and $\hat{g}_{ij}$ to solve for γ^*. As P is very small ($\epsilon \ll 1$), queries are randomly routed to enhance generalization without hurting performance. Let the remaining queries be $Y = Q \setminus P$. In the second stage, the learned weights γ^* are used to route each incoming query $j \in Y$, where each query is directly assigned to the LLM that maximizes the score $(\alpha \hat{d}_{ij} - \gamma_i^* \hat{g}_{ij})$.

Algorithm 1 `PORT`: Routing with Learned γ^*

```
P ← εQ (First ε-frac. of queries)
for j ∈ P do
    Randomly pick w_j ∈ {0} ∪ [M]
    Estimate d̂_ij and ĝ_ij, ∀ i ∈ [M]
    if w_j > 0 then
        Route j to w_j-th LLM
Compute γ* ← arg min_γ F(γ, P)
Y ← Q \ P
for j ∈ Y do
    Estimate d̂_ij and ĝ_ij, ∀ i ∈ [M]
    Compute αd̂_ij − ĝ_ij γ*_i, ∀ i ∈ [M]
    Route j to i = arg max_i(αd̂_ij − ĝ_ij γ*_i)
```

Theoretical Guarantees. Our main theorem shows the proposed algorithm, `PORT`, achieves a competitive ratio close to 1 against the offline optimum C_{opt} under natural assumptions (see Appendix E.1), with proof in Appendix E.6.

Theorem 1 (Informal; see Theorem 2 for formal version)**.** *For any given query set Q with random arrival order, Algorithm 1 satisfies $\frac{C_{alg}}{C_{opt}} \geq 1 - o(1)$ assuming $\frac{C_{opt}}{s_{max}}$ is sufficiently large, where s_{max} is the maximum performance score obtained for any query.*

3 Evaluation

Benchmarks & Baselines & Metrics. We use 3 different benchmarks in our experiments: RouterBench [19], SPROUT [41], and Open LLM Leaderboard v2 [12]. We compare 8 different routing algorithms, classified into two categories: *model-based methods* and *training-free methods*, which are discussed in detail in Appendix Appendix D. Three key metrics are used: (1) *Performance*, the

Table 1: The main results on RouterBench, SPROUT, and Open LLM Leaderboard v2. Here, Perf represents the *Performance*, PPC represents *Performance per Cost*, Tput represents *Throughput*, and RP denotes *Relative Performance* compared with offline approximate optimum ($\hat{C}_{opt}$).

Algorithm	RouterBench					SPROUT					Open LLM Leaderboard v2				
	Perf	Cost	PPC	Tput	RP	Perf	Cost	PPC	Tput	RP	Perf	Cost	PPC	Tput	RP
Random	1384.25	0.427	3243.25	3276	43.10%	2827.6	0.72	3927.29	4742	47.61%	953.0	0.741	1284.37	2877	49.89%
Greedy-Perf	1012.1	0.27	3742.379	1687	31.52%	764.9	0.406	1881.742	1083	12.88%	553.0	0.499	1107.91	1189	28.95%
Greedy-Cost	1626.25	0.46	3534.46	4061	50.64%	3934.7	0.849	4630.41	6789	66.25%	1051.0	0.766	1371.30	3164	55.02%
KNN-Perf	1005.1	0.27	3720.58	1677	31.3%	769.6	0.407	1888.46	1084	12.96%	556.0	0.498	1114.29	1194	29.11%
KNN-Cost	1592.05	0.46	3454.04	4027	49.58%	3905.1	0.85	4593.37	6709	65.75%	991.0	0.766	1293.07	3172	51.88%
BatchSplit	1838.05	0.458	4005.93	3903	57.24%	3975.5	0.83	4784.49	6221	66.94%	1059.0	0.76	1392.07	3099	55.44%
Roberta-Perf	154.5	0.077	2019.00	190	4.81%	458.9	0.283	1621.64	536	7.73%	153.0	0.207	738.21	283	8.01%
Roberta-Cost	481.4	0.129	3738.88	1292	14.99%	3996.2	0.848	4709.22	6765	67.29%	1044.0	0.766	1362.53	3173	54.66%
PORT **(Ours)**	**2718.6**	0.447	**6075.58**	**5195**	**84.66%**	**4513.05**	0.815	**5536.74**	**7475**	**75.99%**	**1465.0**	0.711	**2060.3**	**3692**	**76.7%**
Offline Oracle (Algorithm Upper Bounds Reference)															
Approx Optimum($\hat{C}_{opt}$)	3211.35	0.46	6975.16	6225	100%	5938.99	0.85	6986.45	8781	100%	1910.0	0.765	2493.66	4319	100%
Optimum (C_{opt})	6376.9	0.46	13865.62	6436	198.57%	11934.4	0.848	14060.34	12336	200.94%	4688.0	0.763	6143.64	4688	245.44%

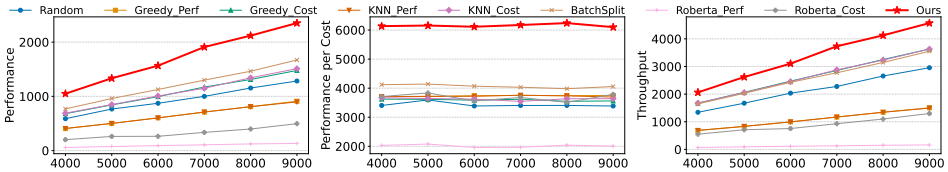

Figure 1: Results on RouterBench with test query volume varying from 4000 to 9000.

total performance score achieved across all queries; (2) *Performance per Cost*, performance-to-cost ratio, reflecting cost efficiency; (3) *Throughput*, the total number of queries processed.

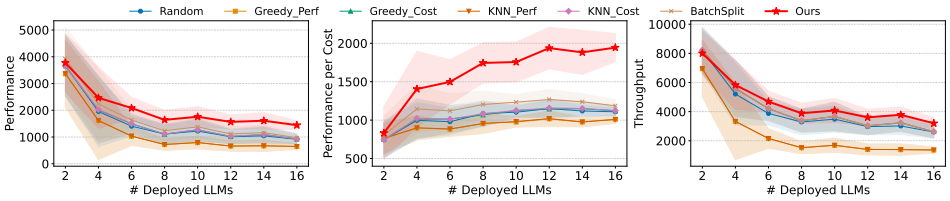

Figure 2: Results on Open LLM Leaderboard v2 when varying LLM deployment configurations.

Main Results. Table 1 reports results on three benchmarks with $\alpha = 0.0001$ and $\epsilon = 0.025$. PORT outperforms all 8 baselines, averaging **3.55**$\times$ better performance, **1.85**$\times$ higher cost efficiency, and **4.25**$\times$ greater throughput. Against the strongest baseline (BatchSplit), it achieves **33%** higher performance, **38%** better cost efficiency, and **24%** higher throughput. It further attains **76-85%** of the offline approximate oracle's performance, consistent with our theoretical guarantees.

Computational & Deployments Scalability. We evaluate the scalability of PORT under two settings: varying query volumes (4k–12k) and varying LLM deployments (2–16 models). As shown in Figures 1 and 2, our method consistently outperforms all baselines across benchmarks and metrics, maintaining the highest performance, cost efficiency, and throughput under both heavy traffic and diverse deployments. Notably, the performance gap widens with load (up to 50% over BatchSplit), underscoring strong robustness and adaptability. More Experiments can be found in Appendix D.

4 Limitations & Conclusion

Limitations. Our algorithm currently targets performance and cost – the two most studied factors [34, 19, 41] – without explicitly modeling other factors. Additional factors may be incorporated by adding constraints to the MILP, whose dual variables naturally extend to new learnable routing weights.

Conclusion. We propose the first efficient, training-free online routing algorithm for high-volume, budget-constrained LLM serving. It leverages ANNS for feature estimation and executes a one-time optimization on a small query set to learn generalizable routing weights. We prove a competitive ratio of $1 - o(1)$ and demonstrate consistent gains across 3 benchmarks and 8 baselines.

References

[1] Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. Automix: Automatically mixing language models. *Advances in Neural Information Processing Systems*, 37:131000–131034, 2024.

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

[3] Lingjiao Chen, M Zaharia, and J Zou. Frugalgpt: how to use large language models while reducing cost and improving performance. arxiv. *arXiv preprint arXiv:2305.05176*, 2023.

[4] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*, 2024.

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

[6] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems, 2021. *URL https://arxiv. org/abs/2110.14168*, 9, 2021.

[7] François Denis. Pac learning from positive statistical queries. In *International conference on algorithmic learning theory*, pages 112–126. Springer, 1998.

[8] Nikhil R Devanur and Thomas P Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *Proceedings of the 10th ACM conference on Electronic commerce*, pages 71–78, 2009.

[9] Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

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

[11] Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. *arXiv preprint arXiv:2410.03834*, 2024.

[12] C. Fourrier, N. Habib, A. Lozovskaya, K. Szafer, and T. Wolf. Open llm leaderboard v2. `https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard`, 2024.

[13] Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Christian Petersen, Alexis Chevalier, and Julius Berner. Mathematical capabilities of chatgpt. *arXiv preprint arXiv:2301.13867*, 2023.

[14] Robert Friel, Masha Belyi, and Atindriyo Sanyal. Ragbench: Explainable benchmark for retrieval-augmented generation systems. *arXiv preprint arXiv:2407.11005*, 2024.

[15] Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA*, volume 8, pages 982–991, 2008.

[16] Surya Narayanan Hari and Matt Thomson. Tryage: Real-time, intelligent routing of user prompts to large language models. *arXiv preprint arXiv:2308.11601*, 2023.

[17] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[18] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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

[20] Qi Huangfu and JA Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018.

[21] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in neural information processing Systems*, 32, 2019.

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

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

[24] Chia-Hsuan Lee, Hao Cheng, and Mari Ostendorf. Orchestrallm: Efficient orchestration of language models for dialogue state tracking. *arXiv preprint arXiv:2311.09758*, 2023.

[25] Kay Lehnert. Ai insights into theoretical physics and the swampland program: A journey through the cosmos with chatgpt. *arXiv preprint arXiv:2301.08155*, 2023.

[26] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023.

[27] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

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

[29] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[30] Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22–es, 2007.

[31] Rui Meng, Ye Liu, Shafiq Rayhan Joty, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. Sfr-embedding-2: Advanced text embedding with multi-stage training, 2024.

[32] Microsoft News Center. Microsoft unveils a 10 k-gpu azure supercomputer for openai, 2020.

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

[34] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data, 2025.

[35] Balaji Rama, Kai Mei, and Yongfeng Zhang. Cerebrum (aios sdk): A platform for agent development, deployment, distribution, and discovery. *arXiv preprint arXiv:2503.11444*, 2025.

[36] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

[37] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[38] Marija Šakota, Maxime Peyrard, and Robert West. 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, 2024.

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

[40] Shubham Singh. Chatgpt statistics (2025): Dau & mau data worldwide, 2025.

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

[42] Zayne Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. Musr: Testing the limits of chain-of-thought with multistep soft reasoning. *arXiv preprint arXiv:2310.16049*, 2023.

[43] Dimitris Stripelis, Zijian Hu, Jipeng Zhang, Zhaozhuo Xu, Alay Dilipbhai Shah, Han Jin, Yuhang Yao, Salman Avestimehr, and Chaoyang He. Tensoropera router: A multi-model router for efficient llm inference. *arXiv preprint arXiv:2408.12320*, 2024.

[44] 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. Challenging big-bench tasks and whether chain-of-thought can solve them, 2022.

[45] Teknium. Openhermes-2.5, 2023.

[46] Hongyi Wang, Felipe Maia Polo, Yuekai Sun, Souvik Kundu, Eric Xing, and Mikhail Yurochkin. Fusing models with complementary expertise. *arXiv preprint arXiv:2310.01542*, 2023.

[47] Xinyuan Wang, Yanchi Liu, Wei Cheng, Xujiang Zhao, Zhengzhang Chen, Wenchao Yu, Yanjie Fu, and Haifeng Chen. Mixllm: Dynamic routing in mixed large language models. *arXiv preprint arXiv:2502.18482*, 2025.

[48] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[49] Kyle Wiggers. Openai ceo sam altman says the company is 'out of gpus', 2025.

[50] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. C-pack: Packaged resources to advance general chinese embedding, 2023.

[51] Haike Xu, Sandeep Silwal, and Piotr Indyk. A bi-metric framework for fast similarity search, 2024.

[52] Weizhe Yuan, Graham Neubig, and Pengfei Liu. Bartscore: Evaluating generated text as text generation. *Advances in neural information processing systems*, 34:27263–27277, 2021.

[53] Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. Large language model cascades with mixture of thoughts representations for cost-efficient reasoning. *arXiv preprint arXiv:2310.03094*, 2023.

[54] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[55] Qiang Zhang, Keyan Ding, Tianwen Lv, Xinda Wang, Qingyu Yin, Yiwen Zhang, Jing Yu, Yuhao Wang, Xiaotong Li, Zhuoyi Xiang, et al. Scientific large language models: A survey on biological & chemical domains. *ACM Computing Surveys*, 57(6):1–38, 2025.

[56] 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. Judging llm-as-a-judge with mt-bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc., 2023.

[57] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560, 1997.

A Background and Formal Problem Setup

LLM Routing. Existing LLM routing research primarily falls into two paradigms. The first focuses on improving response quality while managing cost, typically through ensembling outputs from different LLMs [22, 46] or using cascading strategies that query LLMs sequentially based on their capabilities [3, 1, 53, 24]. For instance, LLM-Blender [22] proposes an ensembling framework that combines outputs from multiple open-source LLMs and selects the optimal response. [46]using outputs Wang et al. [46] propose to fuse outputs of expert models that capture complementary aspects of the data distribution, to generate the final answer. Frugal-GPT [3] adopts a sequential querying strategy, invoking LLMs in order of increasing capability until a satisfactory response is obtained. AutoMix [1] first uses a smaller model to self-verify the quality of the response, based on which it dynamically selects a suitably sized model for handling the query. Although these approaches can improve performance, they incur high latency and computational cost due to multiple model invocations per query.

The second paradigm leverages historical or auxiliary datasets to estimate the performance and cost of each query and select the optimal LLM accordingly [34, 41, 19, 10, 28, 23, 39, 43, 11, 16, 47]. In this line of work, several approaches train model-based predictors using these labeled historical datasets. For example, RouterLLM [34] trains a BERT-base or causal LLM on the historical data annotated with human preferences to enhance routing accuracy. CARROT [41] trains Roberta-base models to estimate both performance and cost for each query, enabling selection of the optimal LLM under any user-defined trade-off between quality and cost. HybridLLM [10] adopts the synthetic preference labels from the MixInstruct dataset [22] based on BARTScore [52], and trains a single BERT-based router for query routing. Zooter [28] distills reward signals from training queries and applies tag-based label enhancement to train a routing function that facilitates the expert selection. TensorOpera [43] introduces a soft label-based strategy based on the BERT similarity scores to train the BERT-based predictor for query routing. GraphRouter [11] constructs a heterogeneous graph to capture the contextual relationship between the query requirements and the LLM capabilities for informed routing decisions. While these training-based approaches are effective for certain LLM deployment settings, they introduce nontrivial training overhead. Furthermore, adapting them to varying LLM deployment configurations typically requires retraining, which makes them unsuitable for resource-constrained online routing with diverse LLM deployment configurations.

Some methods avoid training by using approaches such as KNN [19] or similarity-weighted (SW) ranking [34] to estimate the performance and cost based on the historical data. While these approaches avoid model training overhead, they still incur substantial computational overhead and latency, as they rely on brute-force comparisons against the entire historical dataset to retrieve similar examples or compute similarity scores without any optimization. Specifically, they operate with a high search complexity of $O(N)$, where N is the size of the dataset. As a result, these methods are difficult to scale and impractical for high-volume, budget-constrained online routing scenarios.

Recent efforts [38, 33, 35] have explored formulating LLM routing as MILP. For instance, Sakota et al. [38] introduce two MILP-based strategies, targeting performance-oriented and cost-oriented, respectively. However, these methods face significant challenges in high-volume online routing settings, where queries arrive sequentially, rather than simultaneously, making the offline optimal infeasible to compute.

We complement these works by introducing the first training-free and efficient online routing method with provable performance guarantees, tailored for high-volume, budget-constrained, and dynamic LLM-serving environments.

Problem Definition. Consider an LLM-serving system deployed with M types of LLMs. Each LLM type, indexed by $i \in [M] = \{1, \ldots, M\}$, is allocated a running token budget B_i (i.e., representing the limited token budgets available per time unit), with a total budget equaling B. Typically, the split of the budget is predetermined before the operation of the system. During each time unit, the system receives a set of queries from various end users, denoted by Q. The goal is to design an online routing strategy $x(\cdot)$ that assigns incoming queries in Q to the available LLMs in a way that maximizes the overall quality of responses, under the budget constraints.

The offline version of this problem can be naturally formulated as the following MILP (Objective 7),

$$
\begin{aligned}
\max \sum_{j \in Q} \sum_{i \in [M]} & d_{ij} x_{ij} \\
\text{s.t.} \sum_{j} & g_{ij} x_{ij} \leq B_i \ \text{ for all } i, \\
\sum_{i} & x_{ij} \leq 1 \ \text{ for all } j, \\
& x_{ij} \in \{0, 1\}
\end{aligned} \tag{7}
$$

where d_{ij} denotes the *performance score* of the response from LLM i to query j, and g_{ij} represents the *token budget consumption* for LLM i processing query j (representing the cost). Specifically, g_{ij} can be further decomposed as $f_i^I \cdot \text{len}(j) + f_i^O \cdot \text{len}(a_{ij})$, where $f_i^I \in \mathbb{R}^M$ and $f_i^O \in \mathbb{R}^M$ represent fixed costs per token during the prefill and decoding stages for LLM i, respectively.

However, solving Objective 7 (or a natural relaxation) directly in practical online settings poses several non-trivial challenges: **(i) Inaccessible ground-truth performance and cost:** For any query j, the true performance score d_{ij} and cost g_{ij} are unavailable without accessing the actual LLMs. **(ii) Sequential query arrival under uncertainty:** In practice, queries arrive sequentially rather than simultaneously and must be routed without knowledge of future queries. **(iii) Computational scalability:** High query volume demands routing decisions with low latency and minimal computational overhead. However, methods that rely on solving large-scale MILP or executing computationally intensive model predictors for each incoming query may violate these constraints. **(iv) Deployment scalability:** LLM deployment configurations, such as M or the underlying LLMs, may vary across different environments. Thus, the algorithm must be adaptive to these variations while minimizing adaptation overhead.

Motivated by these challenges, we study the following online routing algorithmic problem: Given a larger set of queries Q arriving in a random sequential order, and a predefined token budget constraint, can we design an online routing algorithm that still achieves a near-optimal cumulative performance? Formally, we aim to ensure $\frac{C_{alg}}{C_{opt}} \geq 1 - o(1)$, where we define $C_{alg} := \sum_j \sum_i d_{ij} \hat{x}_{ij}$ [1] denotes the total performance of the algorithm with routing results $\hat{x}_{ij}$, and $C_{opt} := \sum_j \sum_i d_{ij} x_{ij}^*$ denotes the offline MILP optimum with the optimal solution x_{ij}^*.

B Methodology

To tackle the challenges, our algorithm estimates the performance scores and costs efficiently using a historical dataset (Section B.1), and learns a routing strategy from a small subset of observed queries (of size $\epsilon|Q|$) to guide future query routing (Section B.2).

B.1 Efficient Performance and Cost Estimation

Our solution to estimate d and g is to leverage a historical dataset D with a specialized data structure that supports efficient similarity-based retrieval. Specifically, let $D = \{j, a_j, d_j, g_j\}_{j=1}^n$, where $a_j \in \mathbb{R}^M$ represents the response vector generated by the M LLMs for query j, and $d_j, g_j \in \mathbb{R}^M$ denotes the corresponding performance scores and costs.

For any incoming query $j \in Q$, we apply a classic ANNS method over D in the embedding space, such as DiskANN [21] and HWSN [29], to select the most similar data points for estimation.

This yields a set of approximate nearest neighbors for j, denoted by $R_j \subset D$. The estimated performance score $\hat{d}_{ij}$ and cost $\hat{g}_{ij}$ for LLM i are then computed as the mean over these neighbors: $\hat{d}_{ij} = \frac{1}{|R_j|} \sum_{q \in R_j} d_{iq}, \ \hat{g}_{ij} = \frac{1}{|R_j|} \sum_{q \in R_j} g_{iq}$.

One significant advantage of using ANNS compared with other training-based approaches is the ease of updating the historical dataset D to adopt various deployment configurations without requiring

[1] For simplicity, we use the notation $\sum_j$ to denote $\sum_{j \in Q}$ and $\sum_i$ to denote $\sum_{i \in [M]}$, unless stated otherwise.

model retraining. This is particularly advantageous for LLM-serving providers, as it incurs almost no additional deployment overhead. Online systems can naturally collect and record diverse user queries, which can be directly used to maintain and expand the dataset D. Although collecting ground-truth performance scores and costs typically requires human or automated evaluation, many providers already integrate such mechanisms (e.g., quality feedback from users) in their systems [4]. Furthermore, widely used ANNS algorithms in practice, such as HWSN[2], use a graph-based indexing structure, making it substantially more efficient in search complexity (typical $O(\log |D|)$ in practice) compared to traditional instance-based methods like exact K-Nearest Neighbor (KNN, $O(|D|)$) [19], achieving significant speedup in search time.

B.2 Online Routing from Observed Queries

With the approximate features $\hat{d}_{ij}$ and $\hat{g}_{ij}$, we propose an online routing algorithm in Algorithm 1.

Approximate LP with Control Parameter. The first step is to approximate the original MILP using the estimated features $\hat{d}_{ij}$ and $\hat{g}_{ij}$. In addition, we introduce a control parameter $\alpha > 0$ into the objective, leading to the formulation in Equation (8). The inclusion of α does not affect the optimal solution structure – it simply scales the objective value. In fact, α acts as a control parameter to ensure generalizability, which is discussed in Appendix C.

$$\begin{aligned} \max \sum_{j\in Q}\sum_{i\in[M]} \alpha \hat{d}_{ij} x_{ij} \\ \text{s.t.} \sum_j \hat{g}_{ij} x_{ij} \le B_i,\ \forall i, \\ \sum_i x_{ij} \le 1,\ \forall j, \\ x_{ij} \in \{0,1\},\ \forall i,j \end{aligned} \tag{8}$$

$$\begin{aligned} \max \sum_{j\in Q}\sum_{i\in[M]} \alpha \hat{d}_{ij} x_{ij} \\ \text{s.t.} \sum_j \hat{g}_{ij} x_{ij} \le B_i,\ \forall i, \\ \sum_i x_{ij} \le 1,\ \forall j, \\ x_{ij} \in [0,1],\ \forall i,j \end{aligned} \tag{9}$$

$$\begin{aligned} \min \sum_{i\in[M]} \gamma_i B_i + \sum_{j\in Q} \beta_j \\ \text{s.t.}\ \beta_j \ge \alpha \hat{d}_{ij} - \hat{g}_{ij}\gamma_i,\ \forall i,j, \\ \gamma_i \ge 0, \quad \beta_j \ge 0,\ \forall i,j \end{aligned} \tag{10}$$

Dual LP under Relaxation. Let s_{max} denote the max performance score across all queries, and let $\hat{C}_{opt}$ be the *offline approximate optimum* obtained by solving Equation (8) with α removed. When the ratio $\hat{C}_{opt}/s_{max}$ (or C_{opt}/s_{max}) is sufficiently large as $|Q|$ grows, which commonly holds in practice, particularly in high-volume settings, the optimal value to the relaxed LP closely approximates that of the original MILP. [3] This is formalized in Lemma 3 and 4 with further discussion in E.1. Based on this observation, we apply LP relaxation to Equation (8), allowing x_{ij} to take fractional values in $[0, 1]$. This yields the relaxed approximate LP in Equation (9) with its dual given in Equation (10).

Routing via Learned γ^*. By complementary slackness, if x is the optimal solution to Equation (9) and $(\gamma,\ \beta)$ is optimal solution to its dual, then $x_{ij} > 0 \Leftrightarrow \beta_j = \max_i (\alpha\hat{d}_{ij} - \hat{g}_{ij}\gamma_i)$. This implies that, at optimality, the dual objective can be expressed as a function parameterized by γ: $F(\gamma) = \sum_i \gamma_i B_i + \sum_j \max_i (\alpha\hat{d}_{ij} - \hat{g}_{ij}\gamma_i)$ (11).

Therefore, ideally, given the offline optimal solution γ, each query j should be routed to the LLM i that maximizes $(\alpha\hat{d}_{ij} - \hat{g}_{ij}\gamma_i)$. While it is infeasible to compute such a global offline solution in an online setting, this formalization motivates us to treat γ as a set of routing weights applied across all LLMs to assist the routing process. Building on this insight, we adapt this idea to the online setting: we learn an estimated set of weights γ^* from the first $P = \epsilon|Q|$ queries. The procedure in Algorithm 1 reflects this idea exactly where its first stage is to learn an optimal γ^* that minimizes dual objective $F(\gamma, P)$ over the observed queries: $F(\gamma, P) = \epsilon \sum_i \gamma_i B_i + \sum_{j\in P} \max_i (\alpha\hat{d}_{ij} - \gamma_i\hat{g}_{ij})$ (12).

For each query $j \in P$, the estimated features $\hat{d}_{ij}$ and $\hat{g}_{ij}$ for all models will be calculated and recorded to solve for γ^*. Since this stage requires only a small fraction of the total queries ($\epsilon \ll 1$), random routing is adopted, which may leave some queries in the waiting queue (lines 3, 5-6 in Algorithm 1) to improve the generalizability without degrading overall performance. This approach aligns precisely with the core idea of PAC learning [7, 8], where uniform and unbiased routing ensures the learned weights γ^* generalize to the subsequent routing stage. We formalize this in Lemma 4 and 6. Let the

[2] We note that many other choices are interchangeable here, e.g. see `https://ann-benchmarks.com`

[3] In our experiments, the optimality gap between the relaxed LP and the MILP is only 0.016% on SPROUT

remaining queries be $Y = Q \setminus P$. In the second stage, the learned weights γ^* are used to route each incoming query $j \in Y$. Specifically, each query is directly assigned to the LLM that maximizes the score $(\alpha \hat{d}_{ij} - \gamma_i^* \hat{g}_{ij})$ (line 12 in Algorithm 1). If the selected LLM has exhausted its budget, the query is placed in a queue to await execution.

Compared to existing methods [34, 41, 19], which often involve repeated heavy computations, our algorithm is significantly more efficient. It performs optimization only once over a small subset of queries, making it scalable for deployment in high-volume settings.

C Theoretical Guarantees

In the following main theorem, we show that our algorithm achieves a competitive ratio close to 1 compared to the offline opt C_{opt}, under natural assumptions (see Appendix E.1 for discussion).

Theorem 2. *For any given query set Q with random arrival order, Algorithm 1 satisfies $\frac{C_{alg}}{C_{opt}} \geq 1 - O(\epsilon + \delta)$ assuming $\frac{C_{opt}}{s_{max}} \geq \Omega(\frac{\alpha M \log(M|\Phi|/\epsilon)}{\epsilon^3(1+\delta)})$, where s_{max} is the maximum performance score obtained for any query, and Φ is an ϵ-net defined over all possible routing strategies $x(\gamma)$.*

To establish this theorem, we introduce the following necessary but mild assumption (see Appendix E.1 for a detailed discussion) and auxiliary lemmas. The key intuition behind approximating features of one query j using a different query j' is that the error between their feature values remains bounded, as long as their embedding representations are close. We formalize this as follows:

Assumption 1. *For any query j, j', there exists a value $\eta > 0$ such that if $||\textsc{Emb}(j) - \textsc{Emb}(j')||_2 \leq \eta$, then $\forall\, i \in [M]$, it holds that $(1 - O(\delta))d_{ij'} \leq d_{ij} \leq (1 + O(\delta))d_{ij'}$ and $(1 - O(\delta))g_{ij'} \leq g_{ij} \leq (1 + O(\delta))g_{ij'}$, where $\textsc{Emb}(\cdot)$ represents the embedding function of an embedding model.*

This approximation strategy naturally introduces estimation errors. Thus, it is necessary to quantify the discrepancy between the offline optimum C_{opt} and the offline approximate optimum $\hat{C}_{opt}$. We assume that for any query $j \in Q$, there exists a corresponding feasible set R_j for estimation. This yields the following Lemma 3 (proof in Appendix E.2).

Lemma 3. *Suppose that $\forall j \in Q$, and $\forall j' \in R_j$, we have $||\textsc{Emb}(j) - \textsc{Emb}(j')||_2 \leq \eta$, and that Assumption 1 holds. Then, the offline approximate optimum $\hat{C}_{opt}$ satisfies $\left|\frac{\hat{C}_{opt} - C_{opt}}{C_{opt}}\right| \leq O(\delta)$.*

Algorithm 1 yields the a set of routing results $\hat{x}$, from which we define the *estimated cumulative performance score* as $C_{est} := \sum_j \sum_i \alpha \hat{d}_{ij} \hat{x}_{ij}$. The results $\hat{x}$ are determined by the learned weights γ^* and the budget limitation, and can be represented by $x(\gamma^*)$. Accordingly, C_{est} can be expressed as $C_{est} = \sum_i \min\{E_i, \sum_j \alpha \hat{d}_{ij} x_{ij}(\gamma^*)\}$, where E_i represents the maximum feasible contribution under the budget B_i, defined as: $E_i := \sum_j^k \alpha \hat{d}_{ij} x_{ij}(\gamma^*)$ with $k = \underset{k}{\operatorname{argmax}} \sum_j^k \hat{g}_{ij} x_{ij}(\gamma^*)$, $s.t.\ \sum_j^k \hat{g}_{ij} x_{ij}(\gamma^*) \leq B_i$. We further define the per-LLM estimated performance score $C_{est,i} := \min\{E_i, \sum_j \alpha \hat{d}_{ij} x_{ij}(\gamma^*)\}$. Analyzing C_{est} thus provides a way to evaluate the routing decisions $x(\gamma^*)$. Unfortunately, directly assessing C_{est} is challenging as it has a complex step function with discontinuities. To facilitate analysis, we introduce a relaxed version $\hat{C} := \sum_i \sum_j \alpha \hat{d}_{ij} x_{ij}(\gamma^*)$, which removes the step function present in C_{est}. We further define per-model relaxed estimated performance score $\hat{C}_i := \sum_j \alpha \hat{d}_{ij} x_{ij}(\gamma^*)$, and its counterpart over the observed query set P as $\hat{C}_i(P) := \sum_{j \in P} \alpha \hat{d}_{ij} x_{ij}(\gamma^*)$. Note that for any $j \in P$, we define $x_{ij}(\gamma^*) = 1$ if the index w_j randomly selected in the first stage of the algorithm equals i; otherwise, $x_{ij}(\gamma^*) = 0$.

Definition 1 (**ϵ-Net** [8])**.** Given a parameter $\epsilon > 0$ and routing rule $x(\gamma)$, a set $\Phi \subseteq [0, 1]^M$ is an ϵ-net if, for any $\gamma \in [0, 1]^M$, there exists $\gamma' \in \Phi$ such that $\forall\, i, j,\ |x_{ij}(\gamma) - x_{ij}(\gamma')| \leq \epsilon$.

By removing the discontinuous step function, the analysis becomes tractable. A key requirement for ensuring the performance of the algorithm is the generalizability of estimated routing weights γ^* on the remaining queries Y. To show this, we first prove that the performance disparity between $\hat{C}(P)$ and its expected value $\epsilon\hat{C}$, i.e., $|\hat{C}(P) - \epsilon\hat{C}|$, is bounded. Since directly bounding this gap over all possible values of γ^* is infeasible due to the infinite parameter space, we use an *ϵ-net* [51, 8]

Φ (formally defined in Definition 1) to reduce to a discrete covering set of routing rules, over which we apply a union bound. We then extend the analysis to arbitrary γ^* values by rounding to Φ. The cumulative error over P is bounded in Lemma 4 (proof in Appendix E.3):

Lemma 4. *Let Φ be an ϵ-net and assume that $\frac{C_{opt}}{s_{max}} \geq \Omega(\frac{\alpha M \log(M|\Phi|/\epsilon)}{\epsilon^3(1+\delta)})$. If $\forall\, i$, $|\hat{C}_i(P) - \epsilon\hat{C}_i| \leq z_i$, then $\sum_i z_i \leq O(\epsilon^2\sqrt{(1+\delta)C_{opt}\hat{C}/\alpha})$.*

The following Lemma demonstrates the generalizability of γ^* on remaining queries Y (proof in Appendix E.4). This Lemma highlights the role of the control parameter α in our algorithm: it serves as a bridge to connect the cost and performance.

Lemma 5. *If $\forall i$, it holds that $|\sum_{j\in P}\hat{g}_{ij}x_{ij}(\gamma^*) - \epsilon\sum_j \hat{g}_{ij}x_{ij}(\gamma^*)| \leq O(z_i)$, then there exists a control parameter $\alpha > 0$, such that $\forall i$, (1) $|\hat{C}_i(P) - \epsilon\hat{C}_i| \leq z_i$, and (2) $|\hat{C}_i(P) - \epsilon E_i| \leq O(z_i)$.*

This connection further translates the cost constraint (B) to a performance guarantee, which establishes the generalizability of γ^* to the remaining queries Y, as formalized in Lemma 6.

Lemma 6. *Let Φ be an ϵ-net. If $\frac{C_{opt}}{s_{max}} \geq \Omega(\frac{\alpha M \log(M|\Phi|/\epsilon)}{\epsilon^3(1+\delta)})$, then $C_{est}(Y) \geq (1 - O(\epsilon))C_{est}$.*

Combining the results above, we can prove our main Theorem 2 (proof in Appendix E.6).

D Evaluation

D.1 Experimental Setup

Table 2: Benchmark details for RouterBench, SPROUT, and Open LLM Leaderboard v2 in our experiments.

(a) RouterBench

Dataset	**Size**
MMLU [17]	14042
Hellaswag [54]	10042
GSM8K [6]	7450
ARC Challenge [5]	1470
Winogrande [37]	1267
MBPP [2]	427
MT-Bench [56]	80
Chinese	785
Consensus Summary	362
Bias Detection	285
Test Match	3
Accounting Audit	30
Abstract2title	254
Total	36497

(b) SPROUT

Dataset	**Size**
MATH Lvl 1-5 [18]	9884
MMLU-PRO [48]	11786
GPQA [36]	541
MUSR [42]	748
RAGBench [14]	1827
Openhermes 2.5 [45]	19455
Total	44241
–	–
–	–
–	–
–	–
–	–
–	–
–	–
–	–

(c) Open LLM Leaderboard v2

Dataset	**Size**
MMLU-PRO [48]	12032
MUSR [42]	756
MATH Lvl 1-5 [18]	1324
GPQA [36]	1192
BBH [44]	5761
Total	21065
–	–
–	–
–	–
–	–
–	–
–	–
–	–
–	–

Benchmarks. We evaluate our method on three benchmarks: RouterBench (zero-shot version) [19], SPROUT [41], and Open LLM Leaderboard v2 [12]. All three benchmarks are constructed from multiple dataset sources. Table 2 summarizes the query types in each benchmark, while Table 3 details the LLMs used and their corresponding token costs.

For RouterBench, we identify 13 different data sources in a total of 36497 samples – 6 more than those reported in the original paper – spanning a diverse range of query domains. This benchmark includes 11 different LLMs. Since it does not provide a predefined train/test split, we randomly sample 10000 queries as the test query set and treat the remaining queries as historical data. Note that we do not report per-token costs for models in RouterBench, as [19] does not provide this information.

Table 3: Model lists and token costs ($/1M tokens) for RouterBench(left), SPROUT (middle), and Open LLM Leaderboard v2 (right).

(a) RouterBench

Models
WizardLM-13B-V1.2
claude-instant-v1
claude-v1
claude-v2
gpt-3.5-turbo-1106
gpt-4-1106-preview
code-llama-instruct-34b
llama-2-70b-chat
mistral-7b-chat
mixtral-8x7b-chat
Yi-34B-Chat
–
–
–
–
–
–

(b) SPROUT

Models	Input Cost	Output Cost
claude-3-5-sonnet-v1	3.00	15.00
titan-text-premier-v1	0.50	1.50
openai-gpt-4o	2.50	10.00
openai-gpt-4o-mini	0.15	0.60
granite-3-2b-instruct	0.10	0.10
granite-3-8b-instruct	0.20	0.20
llama-3-1-70b-instruct	0.90	0.90
llama-3-1-8b-instruct	0.20	0.20
llama-3-2-1b-instruct	0.06	0.06
llama-3-2-3b-instruct	0.06	0.06
llama-3-3-70b-instruct	0.90	0.90
mixtral-8x7b-instruct	0.60	0.60
llama-3-405b-instruct	3.50	3.50
–	–	–
–	–	–
–	–	–
–	–	–
–	–	–

(c) Open LLM Leaderboard v2

Models	Cost
Mixtral-8x7B-DPO	0.6
Yi-34B-Chat	0.8
QwQ-32B-Preview	1.2
Qwen2-72B-Instruct	0.9
Qwen2.5-7B-Instruct	0.3
Qwen2.5-72B-Instruct	1.2
WizardLM-2-8x22B	1.2
deepseek-llm-67b-chat	0.9
gemma-2-27b-it	0.8
gemma-2-9b-it	0.3
gemma-2b-it	0.1
Llama-2-13b	0.3
Meta-Llama-3.1-70B	0.9
Mistral-7B-Instruct-v0.1	0.2
Mistral-7B-Instruct-v0.2	0.2
Mistral-7B-Instruct-v0.3	0.2
Mixtral-8x7B-Instruct-v0.1	0.6
Llama-3.1-Nemotron-70B	0.9

Instead, the benchmark includes precomputed costs for each query across all 11 LLMs, which we directly use in our experiments.

For SPROUT, it consists of queries from 6 different datasets, covering different query domains, such as math and RAG. It contains a total of 44241 samples across 13 LLMs. We use the provided training set as the historical dataset, and combine the validation and test sets to construct the test query set.

For Open LLM Leaderboard v2, we follow the same setup and data processing as [41], resulting in 21065 samples. The processed benchmark includes 5 different datasets, with each evaluating different aspects of LLM capabilities. This benchmark uses 18 different LLMs, and we randomly sample 10000 queries as the test set and use the remainder as the historical data. Since most evaluations in Open LLM Leaderboard v2 are likelihood-based, the cost is essentially determined by the length of the input. Following [41], we report the cost per input token in Table 3 and compute the total cost of each query based on input token counts.

In the main setting, the queries are embedded using `bge-base-en-v1.5` [50]. For the diversity consideration, we additionally evaluate with two different embedding models: `SFR-Embedding-2_R` [31] and `gte-Qwen2-1.5B-instruct` [26].

Baselines. We compare 8 different routing algorithms, categorized into two categories: *model-based methods* and *training-free methods*.

For Model-based methods, we follow the training approach in [41] and train two separate Roberta-based models [27] to predict the performance and cost of each query. The performance prediction model is classification-based, aiming to select the optimal LLM with the highest expected performance, while the cost prediction model is regression-based and estimates the expected cost of an incoming query. This results in 2 different baselines:

- **Roberta-perf-routing**, routes each query to the model with the highest predicted performance;
- **Roberta-cost-routing**, routes each query to the model with the greatest available budget.

Training-free methods include 6 baselines:

- **Random routing**, randomly selects a model for each query;
- **Greedy-perf-routing**, uses ANNS to select the model with the highest predicted performance;

Table 4: Average cost and performance of models in RouterBench on historical data.

Model	Cost	Perf	Perf/Cost	$(\text{Perf/Cost})^{0.5}$
WizardLM-13B-V1.2	7.27e–05	0.432	5944	77.10
claude-instant-v1	2.32e–04	0.598	2581	50.80
claude-v1	2.14e–03	0.631	295	17.18
claude-v2	2.41e–03	0.636	264	16.24
gpt-3.5-turbo-1106	2.42e–04	0.617	2546	50.45
gpt-4-1106-preview	3.28e–03	0.781	238	15.43
code-llama-instruct-34b	1.71e–04	0.203	1182	34.38
llama-2-70b-chat	2.02e–04	0.328	1627	40.34
mistral-7b-chat	4.56e–05	0.308	6768	82.27
mixtral-8x7b-chat	1.34e–04	0.550	4098	64.01
Yi-34B-Chat	1.85e–04	0.648	3503	59.19

- **Greedy-cost-routing**, uses ANNS to select the model with the most available budget;
- **KNN-perf-routing** [19], uses KNN to select the model with the highest performance;
- **KNN-cost-routing** [19], uses KNN to select the model with the most available budget;
- **BatchSplit routing**, groups incoming queries into small batches and solves the linear programming (LP) per batch to determine routing.

Note that in the online setting, the true remaining budget of each model is not directly observable during the routing process, as it depends on the actual cost incurred after the query response is generated. Therefore, all cost-based methods and BatchSplit rely on predicted costs, which are obtained from predictive models, ANNS, or KNN, to estimate the remaining available budget and make routing decisions.

We adopt HWSN [29] as the main ANNS algorithm and set the number of candidate neighbors ($|R_j|$) to 5 for both ANNS and KNN. To assess the robustness of our algorithm under varying candidate set sizes, we further vary this number to 3, 7, and 10. However, many alternative methods, such as DiskANN [21], are interchangeable here, as listed in `https://ann-benchmarks.com`. For BatchSplit, we use a mini-batch size of 256 to balance LP computation cost with the low-latency requirements of the practical online routing.

Metrics. In our experiments, we consider three key evaluation metrics: (1) *Performance*, the overall performance score achieved by processing all test queries under the given budget constraints; (2) *Performance per Cost*, the ratio of total performance to the corresponding cost, reflecting overall cost efficiency; (3) *Throughput*, the total number of queries successfully processed, indicating the processing capacity. We evaluate *Throughput* as a key metric because our goal is to design an effective online routing algorithm for high-volume settings with limited token budgets. In these scenarios, the available budget may not suffice to serve all incoming queries within a given time unit. Therefore, *Throughput* reflects how efficiently an algorithm utilizes the available budget to maximize the number of queries successfully served during that current time unit.

Budget. Our algorithm aims to improve routing performance under limited budgets in high-volume settings. To simulate this high-volume, budget-constrained setting, we define the total budget based on the minimal cost required for a single model to process all test queries in the benchmark. In the main setting, the budget is set to this minimal value. To evaluate the robustness of our algorithms, we scale the budget by a factor ranging from 0.25 to 2.

We consider multiple strategies for splitting the total budget across models. In the main setting, we adopt a cost-efficiency-based split strategy. As shown in Table 4, 5, and 6, we report the average performance score, cost, and cost efficiency (Perf/Cost) of each model on the historical data across 3 benchmarks. We observe a substantial disparity in cost efficiency across models. For instance, for SPROUT, the most efficient model achieves a cost efficiency of 9722, while the least efficient only reaches 108, nearly a 100$\times$ difference. Directly allocating the budget based on cost efficiency is thus highly imbalanced, as it can result in nearly all resources being allocated to a single model.

Table 5: Average cost and performance of models in SPROUT on historical data.

Model	Cost	Perf	Perf/Cost	(Perf/Cost)$^{0.5}$
claude-3-5-sonnet-v1	7.65e–03	0.827	108	10.39
titan-text-premier-v1	5.64e–04	0.579	1027	32.04
openai-gpt-4o	4.92e–03	0.846	172	13.11
openai-gpt-4o-mini	3.40e–04	0.808	2378	48.76
granite-3-2b-instruct	8.54e–05	0.553	6473	80.46
granite-3-8b-instruct	1.50e–04	0.659	4403	66.36
llama-3-1-70b-instruct	7.17e–04	0.810	1130	33.62
llama-3-1-8b-instruct	2.43e–04	0.690	2838	53.27
llama-3-2-1b-instruct	6.67e–05	0.460	6904	83.09
llama-3-2-3b-instruct	6.47e–05	0.629	9722	98.60
llama-3-3-70b-instruct	5.52e–04	0.804	1457	38.17
llama-3-405b-instruct	2.01e–03	0.776	385	19.63
mixtral-8x7b-instruct	3.74e–04	0.616	1648	40.59

Table 6: Average cost and performance of models in Open LLM Leaderboard v2 on historical data.

Model	Cost	Perf	Perf/Cost	(Perf/Cost)$^{0.5}$
Yi-34B-Chat	6.57e–04	0.428	652	25.53
Mixtral-8x7B-DPO	4.78e–04	0.401	839	28.97
QwQ-32B-Preview	8.90e–04	0.552	621	24.91
Qwen2-72B-Instruct	6.67e–04	0.562	842	29.02
Qwen2.5-72B-Instruct	8.90e–04	0.561	630	25.10
Qwen2.5-7B-Instruct	2.22e–04	0.420	1887	43.44
WizardLM-2-8x22B	9.85e–04	0.491	499	22.34
deepseek-llm-67b-chat	7.05e–04	0.413	586	24.21
gemma-2-27b-it	6.13e–04	0.462	753	27.43
gemma-2-9b-it	2.30e–04	0.419	1826	42.72
gemma-2b-it	7.66e–05	0.191	2489	49.89
Llama-2-13b	2.47e–04	0.227	919	30.31
Meta-Llama-3.1-70B	6.44e–04	0.548	852	29.18
Mistral-7B-Instruct-v0.1	1.43e–04	0.258	1806	42.50
Mistral-7B-Instruct-v0.2	1.64e–04	0.311	1894	43.52
Mistral-7B-Instruct-v0.3	1.64e–04	0.336	2044	45.21
Mixtral-8x7B-Instruct-v0.1	4.92e–04	0.379	770	27.74
nvidia/Llama-3.1-Nemotron-70B	7.39e–04	0.506	686	26.19

Therefore, we adopt a smoothed version that allocates the budget proportionally to the square root of each model's cost efficiency on the historical data, i.e., according to $(\frac{\text{Perf}}{\text{Cost}})^{0.5}$.

In our robustness evaluations, we explore several alternative budget splitting strategies, including uniform split, random split, extreme split, cost-based split, and performance-based split. In the cost-based split, the budget is allocated inversely proportional to the average cost. However, due to the extreme cost imbalance across models (as also observed in the cost-efficiency-based split), we also adopt a smoothed variant that splits the budget proportional to $(\frac{1}{\text{Cost}})^{0.5}$. In the performance-based split, the budget is directly assigned in proportion to the average performance scores of the models. In the random split, the total budget is randomly allocated across LLMs, and the experiment is repeated 100 times to account for variability. In the extreme split, 80% of the budget is assigned to the h least

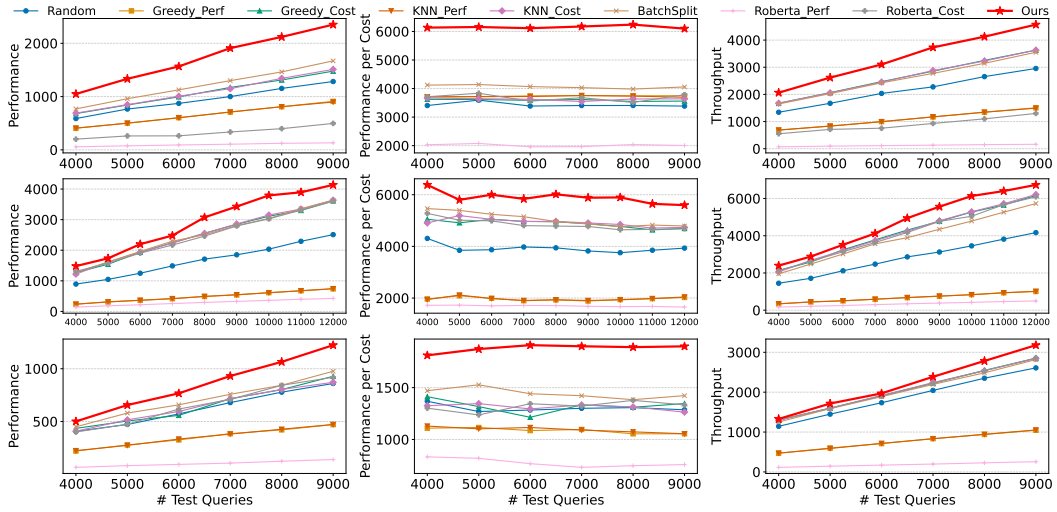

Figure 3: Results with test query volume varying from 4000 to 9000 (12000). Rows correspond to different datasets: RouterBench (top), SPROUT (middle), and Open LLM Leaderboard v2 (bottom).

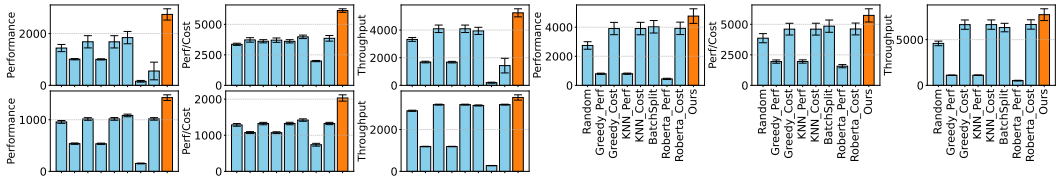

Figure 4: Results under 100 random query orders. (Left to right): the first three subfigures show results on RouterBench, the next three on SPROUT, and the last three on Open LLM Leaderboard v2.

cost-efficient models, and the remaining 20% is uniformly distributed among the others, where h ranges from 1 to 5.

Optimization Implementation. We implement all optimization codes using the CVXPY [9] package. For the one-time optimization step in our algorithm, we use the L-BFGS-B solver [57]. For BatchSplit, which involves solving a linear program for each batch, we adopt the HIGHS solver [20].

Devices. All experiments are conducted on a machine equipped with 16 CPUs and 32GB of memory. For training the Roberta models used in the model-based baselines, we use two NVIDIA H200 GPUs.

D.2 Main Results.

Table 1 presents the main results under the 3 benchmarks, using $\alpha = 0.0001$ and $\epsilon = 0.025$ for our algorithm, with the maximum available test queries, and historical data in the evaluation. Our algorithm consistently outperforms all 8 baselines in performance, cost efficiency, and throughput. On average, it exceeds all baselines by **3.55**$\times$ in performance, **1.85**$\times$ in cost efficiency, and nearly **4.25**$\times$ in throughput. Even against the strongest baseline, BatchSplit, our algorithm still achieves with **33%** higher performance, **38%** better cost efficiency, and **24%** higher throughput across all benchmarks. We further compare our method to the offline approximate oracle ($\hat{C}_{opt}$), and find that it achieves **75.99%** to **84.66%** of the approximate oracle's performance. These results align closely with our theoretical guarantees, further validating the effectiveness of our algorithm.

D.3 Robustness

Query Volume. We vary the number of test queries from 4000 to 12000, serving as different traffic volumes. Figure 3 clearly shows that our method consistently outperforms all baselines across all benchmarks and metrics. It scales gracefully, maintaining the highest performance, cost efficiency, and throughput as the query volume increases from low to extremely high loads. Notably, across all three benchmarks, the performance gap between our method and the strongest baseline (BatchSplit)

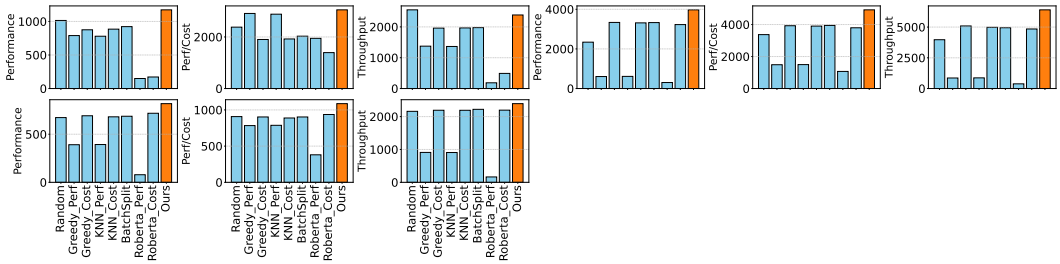

Figure 5: Results under the worst-case query order, where queries are sorted in descending order of cost. (Left to right): the first three subfigures correspond to RouterBench, the next three to SPROUT, and the last three to Open LLM Leaderboard v2.

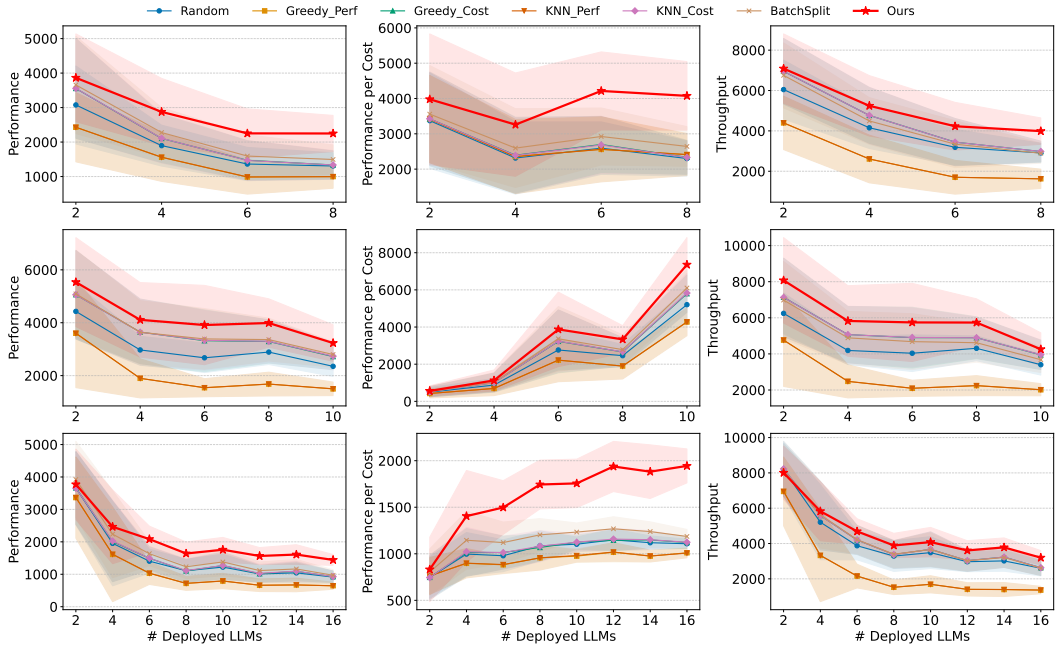

Figure 6: Results when varying the configurations of deployed LLMs. Rows correspond to different datasets: RouterBench (top), SPROUT (middle), and Open LLM Leaderboard v2 (bottom).

widens with increasing load, reaching about 50% higher performance on RouterBench at maximum volume. These results highlight the strong robustness of our approach under varying query volumes.

Query Arrival Order. We evaluate the robustness of our algorithm under varying query arrival orders. Specifically, we consider two settings: (1) a randomized setting, where test queries are independently shuffled 100 times to reflect realistic, unpredictable online environments; and (2) a worst-case setting, where queries are sorted in descending order of their maximum cost across all models. This adversarial order simulates scenarios in which expensive queries arrive early and are more likely to exhaust the budget. Figure 4 shows that our algorithm consistently outperforms all baselines across metrics and benchmarks under random permutations. Even under the worst-case order (Figure 5), our method maintains the advantage, achieving the best performance across all metrics on SPROUT and Open LLM Leaderboard v2, and outperforming all baselines in terms of performance and cost-efficiency on RouterBench. These results demonstrate the robustness of our algorithm towards varying query orders.

Scalability to LLM Deployments. One of the key advantages of our algorithm is its scalability to varying configurations of LLM deployments. To evaluate the robustness of our algorithm, we vary the configurations of deployed LLMs on each benchmark. For RouterBench, which includes 11 distinct LLMs in total, we vary the deployed LLMs from 2 to 8. For SPROUT, which has 13 different

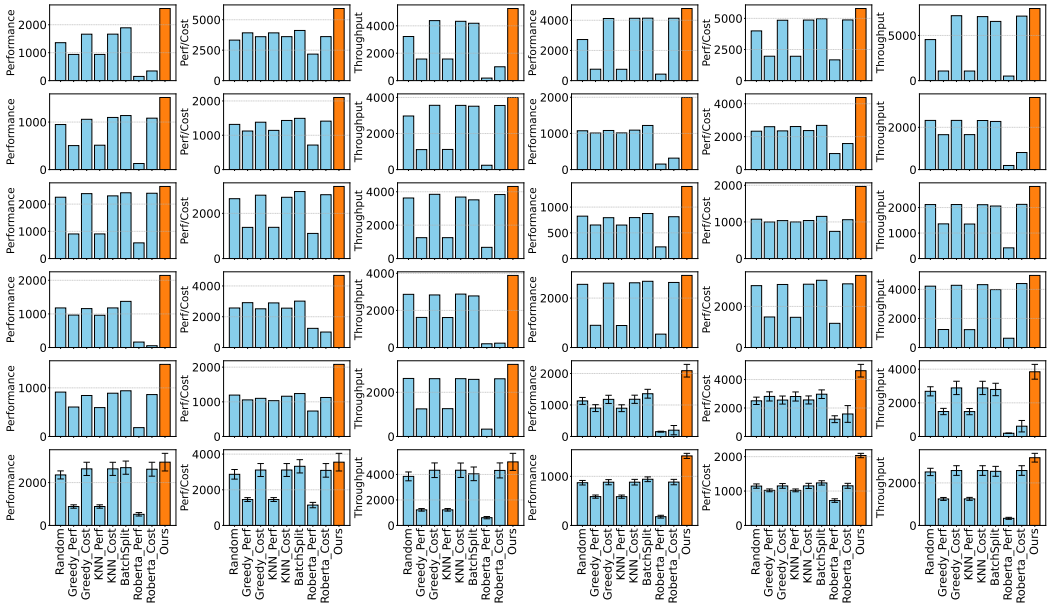

Figure 7: Results under different budget splitting strategies. Each group of 9 subfigures corresponds to one strategy: (a) cost-based split (subfigures 1–9), (b) performance-based split (10–18), (c) uniform split (19–27), and (d) random split (28–36). Within each group, the first three subfigures correspond to RouterBench, the next three to SPROUT, and the last three to Open LLM Leaderboard v2.

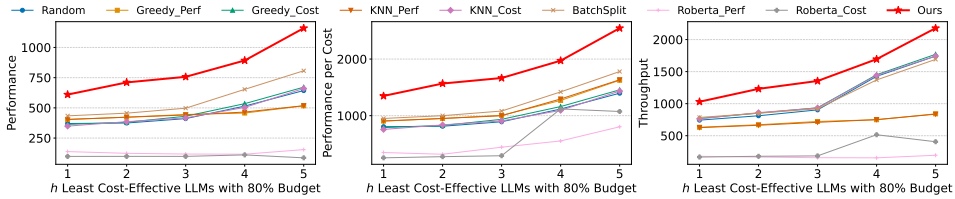

Figure 8: Results on RouterBench under the extreme budget split.

LLMs, we vary the deployed LLMs from 2 to 10. For Open LLM Leaderboard v2, which includes 18 distinct models, we vary the number of deployed LLMs from 2 to 16. In all settings where the number of deployed LLMs is fewer than the maximum, we randomly sample the deployed LLMs and repeat the experiment 10 times to ensure diversity and coverage of possible configurations. Note that in this experiment, we compare only against training-free methods, as retraining models used in model-based methods for each of the numerous settings would incur significant computational and deployment overhead. As shown in Figure 6, our algorithm consistently achieves strong performance across all deployment configurations, where the cost-efficiency gap over other baselines steadily increases. These results demonstrate the robustness and adaptability of our method to diverse and dynamic LLM serving environments.

Budget Split. One key factor affecting the performance of routing algorithms is the choice of budget split strategy. We extend the default cost-efficiency-based split to five alternative strategies. As shown in Figure 7, our algorithm consistently outperforms all baselines across all benchmarks and metrics under four different split strategies: cost-based, performance-based, uniform, and random. Even under the extreme split setting (Figure 9), where a large portion of the budget is concentrated on a few low-efficiency models, our method maintains leading performance. Notably, when 80% of the budget is allocated to a single model ($h = 1$), our algorithm achieves nearly $\mathbf{2}\times$ the performance of the strongest baseline (BatchSplit) on the RouterBench. On Open LLM Leaderboard v2, it also demonstrates approximately $\mathbf{2}\times$ higher cost efficiency than all baselines. These results underscore the strong robustness and adaptability of our algorithm against a wide range of complex and imbalanced budget allocation schemes.

Total Budget. To evaluate the performance of our algorithms in different budget settings, we scale the total budget B from 0.25 to $2\times$ the cost of the cheapest model. Results shown in Figure 10 demonstrate

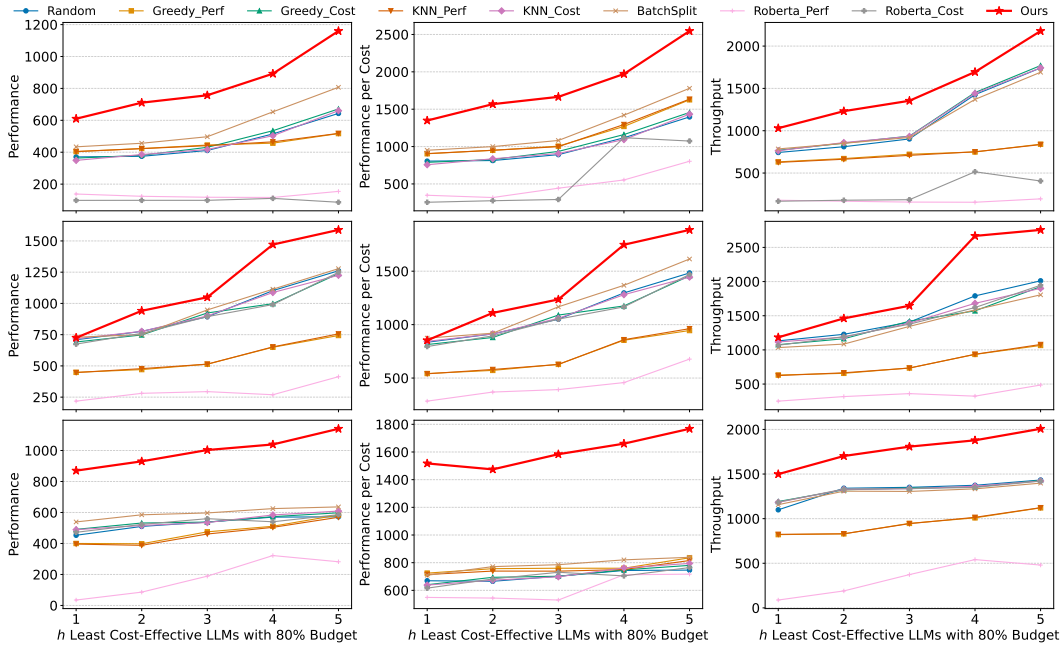

Figure 9: Results under the extreme budget split. Rows correspond to different datasets: RouterBench (top), SPROUT (middle), and Open LLM Leaderboard v2 (bottom).

that our algorithm achieves competitive or superior performance compared to all baselines across benchmarks and metrics. Notably, even under the extremely constrained budget, it maintains a clear advantage. For instance, at $B = 0.25$, it achieves nearly $\mathbf{2}\times$ the performance and cost efficiency of the strongest baseline (BatchSplit) on the Open LLM Leaderboard v2. These results underscore the robustness of our method and its adaptability to different resource availability scenarios.

D.4 Historical Data Size & ANNS/KNN Candidates

Historical Data Size. To evaluate the robustness of our algorithm with respect to the size of historical data, we vary the number of historical data points used in ANNS and KNN from 5000 to 25000. For RouterBench, whose maximum available historical data is 26497, we randomly sample the subset of data points from 5000 to 25000. For RouterBench, which contains up to 26497 historical records, we randomly sample subsets within this range. For SPROUT, with a maximum of 30968 records, we similarly sample subsets from 5000 to 25000. For Open LLM Leaderboard v2, which has 11065 historical records, we sample subsets ranging from 5000 to 10000. Note that in this experiment, training-based methods are trained on the full historical dataset to provide a strong baseline for comparison. As shown in Figure 11, the results clearly show that varying the amount of historical data used in ANNS and KNN has minimal impact on performance, with our algorithm consistently and significantly outperforming all baselines across all settings. This demonstrates the robustness and efficiency of our algorithm: even with as few as 5000 historical records, it remains effective for routing.

Number of ANNS/KNN Candidates. To investigate the impact of the number of search candidates of ANNS and KNN on routing performance, we vary the candidate pool size from the default 5 to 3, 7, and 10. As shown in Figure 12, the results show that performance is marginally affected, with a slight improvement as the candidate pool size increases. Notably, our algorithm consistently maintains a leading position across all settings. Even with as few as 3 search candidates, it outperforms all baselines across benchmarks, demonstrating its stability and robustness to the choice of candidate pool size.

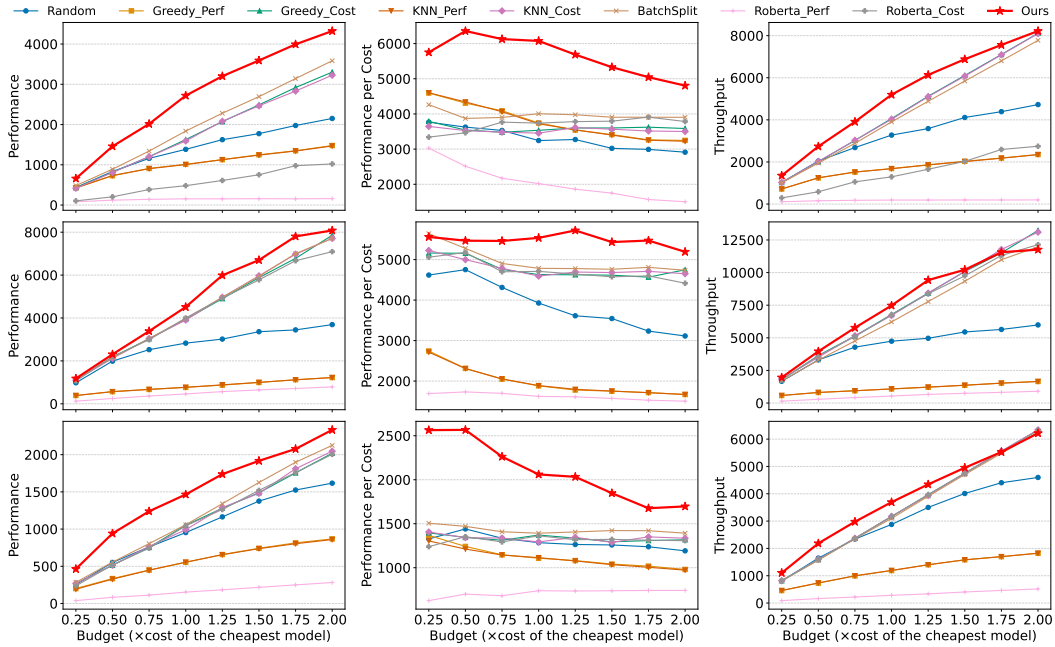

Figure 10: Results when varying total budget B from 0.25 to $2\times$ the cost of the cheapest model. Rows correspond to different datasets: RouterBench (top), SPROUT (middle), and Open LLM Leaderboard v2 (bottom).

D.5 Ablation Study

Impact of embedding models. The choice of embedding model is also important for estimating the performance score and cost of incoming queries. To evaluate the adaptivity of our algorithm to different embedding models used in ANNS, we conduct experiments with two larger embedding models: `SFR-Embedding-2_R` [31] and `gte-Qwen2-1.5B-instruct` [26]. As shown in Figure 13, our algorithm consistently outperforms all baselines across all benchmarks, demonstrating its robustness to the choice of embedding model.

Impact of α and ϵ. We evaluate the sensitivity of our method to the parameters α and ϵ, as shown in Figure 14. For α, we observe a consistent decline in performance as it increases, with the best result achieved at $\alpha = 0.0001$. This trend aligns with Lemma 5, which shows that a larger α amplifies the gap between the performance of our algorithm on the observed subset P, $\hat{C}(P)$, and its expected value $\epsilon\hat{C}$. This increased discrepancy results in greater deviation in performance on future queries, ultimately leading to a reduction in overall performance on the full query set Q. For the parameter ϵ, we observe that increasing its value initially improves performance, reaching a peak at around $\epsilon = 0.025$, after which further increases lead to a decline. This aligns with the intuition that ϵ controls the number of samples used in the learning stage: too few samples lead to underfitting, while too many may lead to overfitting to the observed data.

E Theoretical Proofs

E.1 Discussion on Our Theoretical Assumptions

We introduce several mild assumptions in our theoretical analysis and provide justifications for each below.

Random arrival order. We assume that the queries arrive in a random order, i.e., they can be picked adversarially, but their order is randomly permuted. Note that this is a weaker requirement than assuming the queries are sampled i.i.d. from an unknown distribution. This is because if the queries are sampled i.i.d., then they also satisfy the random order requirement, and thus our assumption is weaker. Furthermore, the random order model is also a popular assumption in many

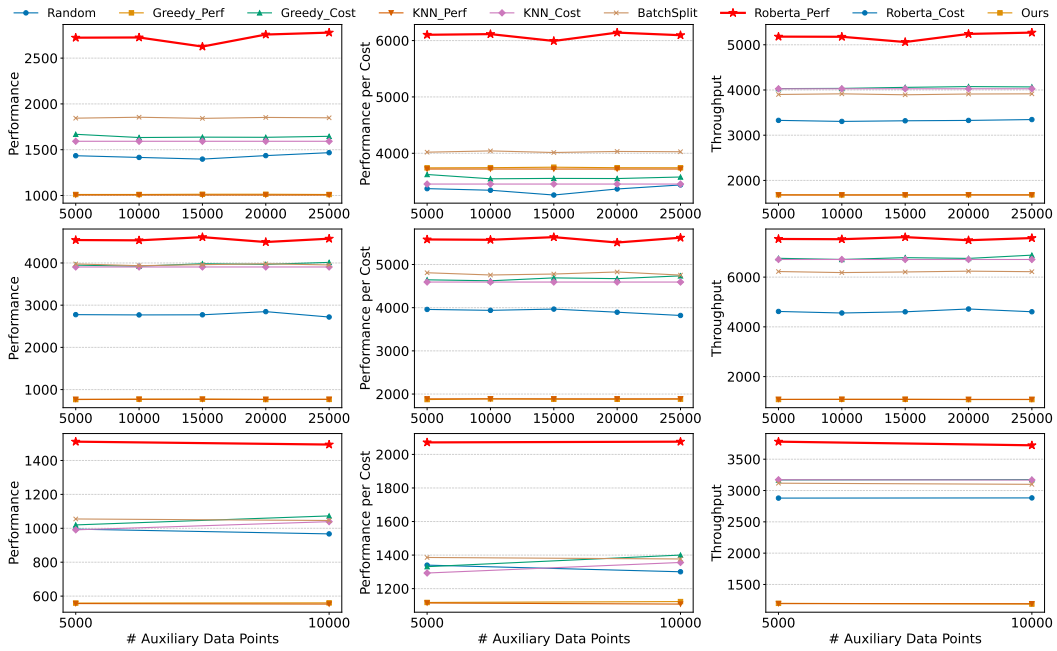

Figure 11: Results when varying the number of historical data points. Rows correspond to different datasets: RouterBench (top), SPROUT (middle), and Open LLM Leaderboard v2 (bottom).

other online algorithms [30, 8], since it typically allows one to go beyond worst-case hardness and obtain meaningful theoretical guarantees. In practice, we perform robustness studies by changing the query order and observe that our algorithm still outperforms baselines; see Appendix D.3.

"C_{opt} is enough large" is practial. In Theorem 2, we assume that $\frac{C_{opt}}{s_{max}} \geq \Omega(\frac{\alpha M \log(M|\Phi|/\epsilon)}{\epsilon^3(1+\delta)})$, which is naturally aligned with high-volume practical settings. This is because C_{opt} depends on the total number of queries $|Q|$, whereas our lower bound assumption scales as a function of M, the number of LLMs. Typically, M can be thought of as a constant while $|Q|$, the total number of queries, should grow over time. Indeed, in all of our benchmarks, we observed that for *every query* j, the optimal solution x^*_{ij} (that is, the performance score achieved by the optimal solution on query j) satisfies $\sum_{i\in[M]} d_{ij}x^*_{ij} \geq$0.462 consistently on Open LLM Leaderboard v2, 0.639 on RouterBench, and 0.887 on SPROUT, meaning that in practice, C_{opt} scales as $\Omega(Q)$ (linear in Q). On the other hand, for constant ϵ and α (e.g. we use $\alpha = 0.0001$, $\epsilon = 0.025$ in our main setting), our required lower bound is a polynomial of M (more specifically, it is close to a quadratic since Φ may depend exponentially in M but it is inside a logarithm). However, we think of M as a constant. For example, $M \leq 18$ in all of our experiments. Even if M is large, it is reasonable to assume that it is orders of magnitude smaller than $|Q|$ (in our experiments, Q is on the order of thousands to tens of thousands, and in real-world systems, it can be much larger).

Our assumption is further corroborated by the relaxation used in Appendix B.1, where if $\frac{\hat{C}_{opt}}{s_{max}}$ is sufficiently large, the original MILP can be closely approximated by its fractional LP relaxation with only a negligible optimality gap. We validate this in our main experiments over three benchmarks with the observed optimality gaps being: (i) 0.016% on SPROUT, (ii) 0.086% on RouterBench, and (iii) 0.3% on Open LLM Leaderboard v2.

Lastly, we remark that similar reasonable lower-bound assumptions have been made in prior work on the online matching problem [15, 8, 30], where it is commonly assumed that bids for the items are small compared to the total budget.

"Estimating query features via similarity" is natural. The core intuition behind most previous literature on predictive LLM routing [19, 34, 47, 41, 43], which trains a model-based predictor on auxiliary datasets to estimate performance and cost of future queries, aligns with Assumption 1. Specifically, these approaches assume that similar queries share similar routing-relevant features. If

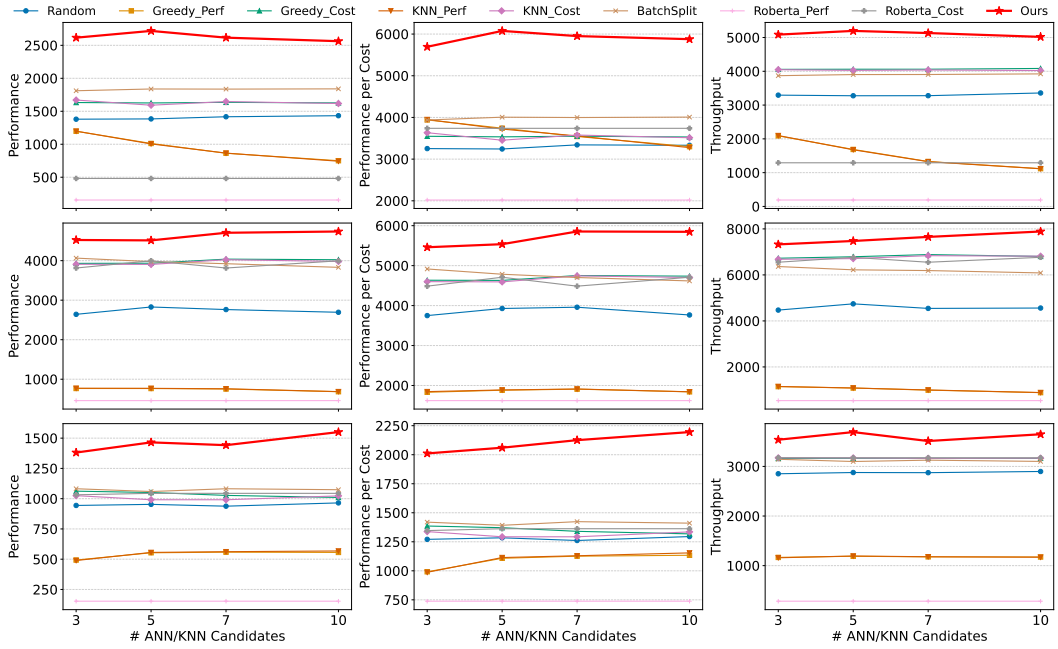

Figure 12: Results when varying the number of search candidates. Rows correspond to different datasets: RouterBench (top), SPROUT (middle), and Open LLM Leaderboard v2 (bottom).

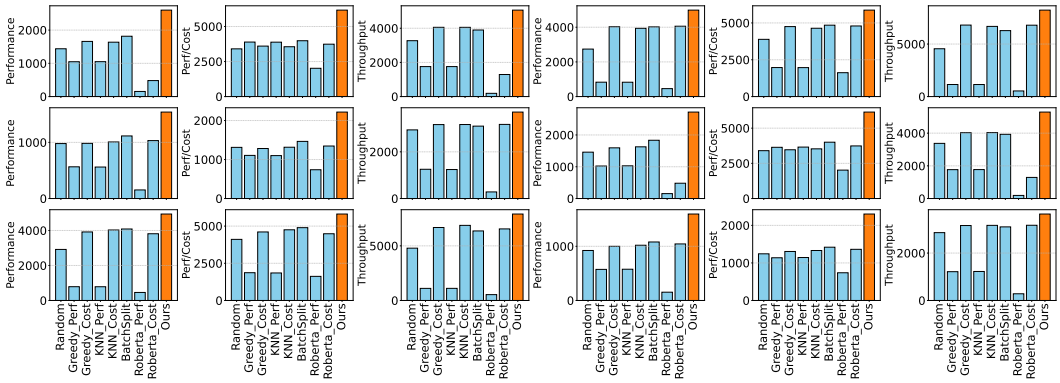

Figure 13: Results using different embedding models. Each group of 9 subfigures corresponds to one embedding: (a) gte-Qwen2-1.5B-instruct (subfigures 1–9), (b) SFR-Embedding-2_R (subfigures 10–18). Within each group, the first three subfigures correspond to RouterBench, the next three to SPROUT, and the last three to Open LLM Leaderboard v2.

no such relationship exists, then historical data would be uninformative, and it would be impossible to leverage historical data for effective future routing. In this case, no meaningful inference can be made without directly querying the LLMs, and routing decisions degenerate to random selection.

One may argue that useful routing information can still be obtained by querying only a subset of LLMs. This idea is echoed in works that adopt cascading strategies, where LLMs are queried sequentially based on their observed capabilities on historical data [1, 3]. However, these approaches also fundamentally rely on a related assumption: LLMs that perform well on historical data or benchmarks are likely to perform well on future queries. Furthermore, these methods introduce significant latency and computational overhead, as they may consume substantial token budgets before reaching the optimal LLM. This makes them impractical for high-volume online routing scenarios with tight token budgets and low-latency requirements.

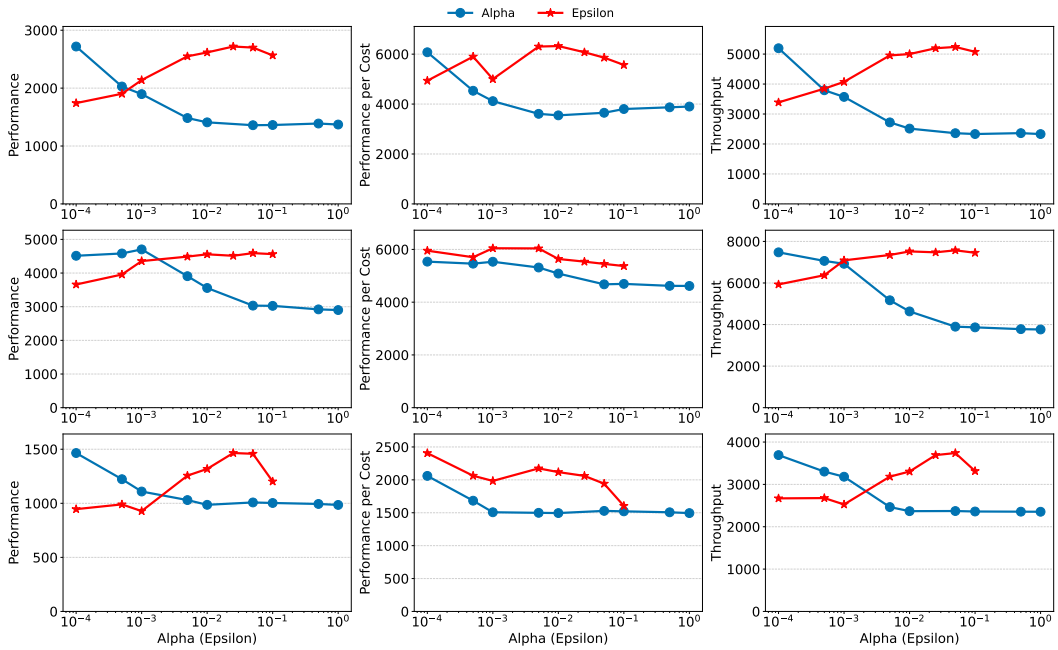

Figure 14: Results when varying α and ϵ. Rows correspond to different datasets: RouterBench (top), SPROUT (middle), and Open LLM Leaderboard v2 (bottom).

If no useful routing information can be inferred from historical data, no matter whether it is related to model capability or query features, then the system would need to query all LLMs to make informed routing decisions. This is because in the worst case, under an adversary query targeting the LLM query order in the system, all LLMs may present extremely low performance, except the last LLM. In this situation, no strategy can be adopted to improve overall routing performance without assessing all LLMs. Therefore, it is important and natural to establish our assumption or use a related notion, e.g., Assumption 1, that links historical information with future queries to obtain meaningful routing information and improve routing performance.

E.2 Proof of Lemma 3

Lemma 2. *Suppose that* $\forall j \in Q$, *and* $\forall j' \in R_j$, *we have* $||\textsc{Emb}(j) - \textsc{Emb}(j')||_2 \leq \eta$, *and that Assumption 1 holds. Then, the offline approximate optimum* $\hat{C}_{opt}$ *satisfies* $\left|\frac{\hat{C}_{opt} - C_{opt}}{C_{opt}}\right| \leq O(\delta)$.

Proof. Let x_1 denote the optimal solution to C_{opt}, and x_2 the optimal solution to $\hat{C}_{opt}$.

Case 1: $C_{opt} \geq \hat{C}_{opt}$. We have

$$
\begin{aligned}
\left|\frac{\hat{C}_{opt} - C_{opt}}{C_{opt}}\right| &= \left|\frac{\sum_j \sum_i \hat{d}_{ij} x_{2ij} - \sum_j \sum_i d_{ij} x_{1ij}}{\sum_j \sum_i d_{ij} x_{1ij}}\right| \\
&\leq \left|\frac{\sum_j \sum_i (\hat{d}_{ij} - d_{ij}) x_{1ij}}{\sum_j \sum_i d_{ij} x_{1ij}}\right| \\
&= \left|\frac{\sum_j \sum_i (\frac{1}{|R_j|} \sum_{q \in R_j} d_{ik} - d_{ij}) x_{1ij}}{\sum_j \sum_i d_{ij} x_{1ij}}\right| \\
&\leq \left|O(\delta) \frac{\sum_j \sum_i d_{ij} x_{1ij}}{\sum_j \sum_i d_{ij} x_{1ij}}\right| \\
&\leq O(\delta)
\end{aligned}
$$

where the second inequality follows from the fact that $\sum_j \sum_i \hat{d}_{ij} x_{1ij} \leq \sum_j \sum_i \hat{d}_{ij} x_{2ij}$ when x_{2ij} is the optimal solution to $\hat{C}_{opt}$, and the fourth inequality follows Assumption 1.

Case 2: $C_{opt} < \hat{C}_{opt}$. We have

$$\begin{aligned}
\left|\frac{\hat{C}_{opt} - C_{opt}}{C_{opt}}\right| &= \left|\frac{\sum_j \sum_i \hat{d}_{ij} x_{2ij} - \sum_j \sum_i d_{ij} x_{1ij}}{\sum_j \sum_i d_{ij} x_{1ij}}\right| \\
&\leq \left|\frac{\sum_j \sum_i (\hat{d}_{ij} - d_{ij}) x_{2ij}}{\sum_j \sum_i d_{ij} x_{1ij}}\right| \\
&= \left|\frac{\sum_j \sum_i (\frac{1}{|R_j|} \sum_{q \in R_j} d_{ik} - d_{ij}) x_{2ij}}{\sum_j \sum_i d_{ij} x_{1ij}}\right| \\
&\leq \left|O(\delta) \frac{\sum_j \sum_i d_{ij} x_{2ij}}{\sum_j \sum_i d_{ij} x_{1ij}}\right| \\
&\leq O(\delta)
\end{aligned}$$

where the second inequality uses the fact that $\sum_j \sum_i d_{ij} x_{2ij} \leq \sum_j \sum_i d_{ij} x_{1ij}$ when x_{1ij} is the optimal solution to C_{opt}, and the fourth inequality follows Assumption 1 and the final inequality follows $\sum_j \sum_i d_{ij} x_{2ij} \leq \sum_j \sum_i d_{ij} x_{1ij}$. □

E.3 Proof of Lemma 4

Lemma 3. *Let Φ be an ϵ-net and assume that*

$$\frac{C_{opt}}{s_{max}} \geq \Omega\left(\frac{\alpha M \log(M|\Phi|/\epsilon)}{\epsilon^3(1+\delta)}\right).$$

If $\forall\, i$, $|\hat{C}_i(P) - \epsilon \hat{C}_i| \leq z_i$, then

$$\sum_i z_i \leq O\left(\epsilon^2 \sqrt{(1+\delta) C_{opt} \hat{C} / \alpha}\right)$$

Proof. Inspired by the techniques used in [8, 30, 15], we first prove that, under the given setting, the partial performance score can be accurately estimated with high probability. Specifically, for each $i \in [M]$ and $\gamma \in \Phi$, we evaluate the following probability

$$\Pr(|\hat{C}_i(P) - \epsilon \hat{C}_i| > z_i).$$

Observe that, for any $j \in P$, the variance satisfies $\textsc{Var}[\alpha \hat{d}_{ij} x_{ij}(\gamma^*)] \leq \frac{1}{|Q|} ||\hat{C}_i||_2^2$, which implies $\sum_{j \in P} \textsc{Var}[\alpha \hat{d}_{ij} x_{ij}(\gamma^*)] \leq \epsilon ||\hat{C}_i||_2^2$. By applying Bernstein's inequality, we obtain

$$\Pr(|\hat{C}_i(P) - \epsilon \hat{C}_i| > z_i) \leq 2 \exp\left(-\frac{z_i^2/2}{\epsilon ||\hat{C}_i||_2^2 + \frac{z_i}{3} \alpha s_{max}}\right)$$

We aim to find such z_i that this probability is below a given tolerance τ. Set RHS as τ and solve for z_i, we obtain

$$z_i = \frac{\frac{2}{3} \log \frac{2}{\tau} \alpha s_{max} + \sqrt{(\frac{2}{3} \log \frac{2}{\tau} \alpha s_{max} + 8\epsilon \log \frac{2}{\tau} ||\hat{C}_i||_2^2}}{2} \leq \frac{2}{3} \log \frac{2}{\tau} \alpha s_{max} + \sqrt{2\epsilon \log \frac{2}{\tau} ||\hat{C}_i||_2^2}$$

Without affecting the analysis, we define $z_i = \frac{2}{3} \log \frac{2}{\tau} \alpha s_{max} + \sqrt{2\epsilon \log \frac{2}{\tau} ||\hat{C}_i||_2^2}$ for simplicity.

Set $\tau = \epsilon/(M|\Phi|)$, and use $\frac{C_{opt}}{s_{max}} \geq \Omega(\frac{\alpha M \log(M|\Phi|/\epsilon)}{\epsilon^3(1+\delta)})$, we have

$$\begin{aligned}
\sum_i z_i &= \sum_i \frac{2}{3} \log \frac{2}{\tau} \alpha s_{max} + \sum_i \sqrt{2\epsilon \log \frac{2}{\tau} ||\hat{C}_i||_2^2} \\
&\leq O(M \log \frac{1}{\tau} \alpha s_{max}) + \sum_i \sqrt{2\epsilon \log \frac{2}{\tau} \hat{C}_i s_{max}} \\
&\leq O(\epsilon^3 C_{opt}(1+\delta)) + \sqrt{2M\epsilon \log \frac{2}{\tau} \hat{C} s_{max}} \\
&\leq O(\epsilon^3 C_{opt}(1+\delta)) + O(\epsilon^2 \sqrt{(1+\delta)}) \sqrt{C_{opt}\hat{C}/\alpha} \\
&\leq O\left(\epsilon^2 \sqrt{(1+\delta) C_{opt} \hat{C}/\alpha}\right)
\end{aligned}$$

This completes the proof. □

E.4 Proof of Lemma 5

Lemma 4. *If $\forall i$, it holds that $|\sum_{j\in P} \hat{g}_{ij} x_{ij}(\gamma^*) - \epsilon \sum_j \hat{g}_{ij} x_{ij}(\gamma^*)| \leq O(z_i)$, then there exists a control parameter $\alpha > 0$, such that $\forall i$, (1) $|\hat{C}_i(P) - \epsilon \hat{C}_i| \leq z_i$, and (2) $|\hat{C}_i(P) - \epsilon E_i| \leq O(z_i)$.*

Proof. Define $M_i := \sum_{j\in P} \hat{d}_{ij} x_{ij}(\gamma^*) - \epsilon \sum_j \hat{d}_{ij} x_{ij}(\gamma^*)$ and $H_i := \sum_{j\in P} \hat{d}_{ij} x_{ij}(\gamma^*) - \epsilon \sum_j^k \hat{d}_{ij} x_{ij}(\gamma^*)$. Then, we aim to ensure the following two conditions hold: (1) $\alpha|M_i| \leq z_i$, and (2) $\alpha|H_i| \leq O(z_i)$.

If $M_i = 0$, condition (1) holds trivially for any $\alpha > 0$; similarly, if $H_i = 0$, condition (2) also holds for any $\alpha > 0$.

Now consider the case where $M_i \neq 0$, then it requires that $\alpha \leq \frac{z_i}{|M_i|}$, and when $H_i \neq 0$, we need to have $\alpha \leq \frac{O(z_i)}{|H_i|}$. Let I be the set of indice i for which $H_i \neq 0$ or $M_i \neq 0$. For each $i \in I$, we have

$$V_{i,1} = \begin{cases} \frac{z_i}{|M_i|} & \text{if } M_i \neq 0 \\ \infty & \text{if } M_i = 0 \end{cases} \qquad V_{i,2} = \begin{cases} \frac{O(z_i)}{|H_i|} & \text{if } H_i \neq 0 \\ \infty & \text{if } H_i = 0 \end{cases}$$

Let $V_i := \min\{V_{i,1}, V_{i,2}\}$, and set

$$\alpha = \inf_{i\in I}\{V_i\}$$

Since $V_i > 0$ for all $i \in I$, it follows that $\alpha > 0$. Therefore, we find a control parameter $\alpha > 0$ such that both conditions (1) and (2) are satisfied. □

E.5 Proof of Lemma 6

Lemma 5. *Let Φ be an ϵ-net. If $\frac{C_{opt}}{s_{max}} \geq \Omega(\frac{\alpha M \log(M|\Phi|/\epsilon)}{\epsilon^3(1+\delta)})$, then $C_{est}(Y) \geq (1 - O(\epsilon))C_{est}$.*

Proof. We first prove that $\forall\, i$, it holds that $\max\{\hat{C}_i, F_i(\gamma^*)\} - C_{est,i} \leq \frac{O(z_i)}{\epsilon}$ where

$$F_i(\gamma^*) := \gamma_i^* B_i + \sum_j (\alpha \hat{d}_{ij} - \gamma_i^* \hat{g}_{ij}) x_{ij}(\gamma^*)$$

From Equation (11), it follows immediately that $\sum_i F_i(\gamma^*) = F(\gamma^*)$.

Case 1: $\gamma_i^* > 0$. In this case,

$$\max\{\hat{C}_i, F_i(\gamma^*)\} - C_{est,i} = \max\{\hat{C}_i, \hat{C}_i + \gamma_i^*(B_i - \sum_j \hat{g}_{ij} x_{ij}(\gamma^*))\} - \min\{E_i, \hat{C}_i\}$$

which follows $C_{est,i} = \min\{E_i, \hat{C}_i\}$. We consider two sub-cases separately:

Sub-case 1.1: $\sum_j \hat{g}_{ij} x_{ij}(\gamma^*) \leq B_i$.

Then,

$$\max\{\hat{C}_i, F_i(\gamma^*)\} - C_{est,i} = \gamma_i^*(B_i - \sum_j \hat{g}_{ij} x_{ij}(\gamma^*)) \leq B_i - \sum_j \hat{g}_{ij} x_{ij}(\gamma^*)$$

Due to complementary slackness conditions over observed queries P, we have $\sum_{j \in P} \hat{g}_{ij} x_{ij}(\gamma^*) = \epsilon B_i$. Thus, it follows that

$$\begin{aligned} &|\sum_{j \in P} \hat{g}_{ij} x_{ij}(\gamma^*) - \epsilon \hat{g}_{ij} x_{ij}(\gamma^*)| = \epsilon |B_i - \hat{g}_{ij} x_{ij}(\gamma^*)| \leq O(z_i) \\ \Rightarrow &\max\{\hat{C}_i, F_i(\gamma^*)\} - C_{est,i} \leq \frac{O(z_i)}{\epsilon} \end{aligned}$$

Sub-case 1.2: $\sum_j \hat{g}_{ij} x_{ij}(\gamma^*) > B_i$.

We obtain,

$$\max\{\hat{C}_i, F_i(\gamma^*)\} - C_{est,i} = \hat{C}_i - E_i$$

From Lemma 5, it can be expanded and bounded as:

$$\begin{aligned} \hat{C}_i - E_i &= \frac{1}{\epsilon}(\sum_j \epsilon \alpha \hat{d}_{ij} x_{ij}(\gamma^*) - \epsilon E_i) \leq \frac{O(z_i)}{\epsilon} \\ \Rightarrow &\max\{\hat{C}_i, F_i(\gamma^*)\} - C_{est,i} \leq \frac{O(z_i)}{\epsilon} \end{aligned}$$

Case 2: $\gamma_i^* = 0$. Then,

$$\max\{\hat{C}_i, F_i(\gamma^*)\} - C_{est,i} = \hat{C}_i - \min\{\hat{C}_i, E_i\}$$

When $\sum_j \hat{g}_{ij} x_{ij}(\gamma^*) \leq B_i$, this difference $\hat{C}_i - \min\{\hat{C}_i, E_i\} = 0$. Thus, we only consider the situation $\sum_j \hat{g}_{ij} x_{ij}(\gamma^*) > B_i$, leading to

$$\max\{\hat{C}_i, F_i(\gamma^*)\} - C_{est,i} = \hat{C}_i - \min\{\hat{C}_i, E_i\} \leq \hat{C}_i - E_i \leq \frac{O(z_i)}{\epsilon}$$

which follows Lemma 5.

Lower Bound on $C_{est}(Y)$. From Lemma 4 and Lemma 5, we have,

$$\begin{aligned} C_{est,i}(Y) &= \min\{\hat{C}_i - \hat{C}_i(P), E_i - \sum_{j \in P} \alpha \hat{d}_{ij} x_{ij}(\gamma^*)\} \\ &\geq \min\{(1-\epsilon)\hat{C}_i - z_i, (1-\epsilon)E_i - O(z_i)\} \\ &\geq (1-\epsilon)C_{est,i} - O(z_i) \end{aligned}$$

where second inequality follows that $|\hat{C}_i(Y) - (1-\epsilon)\hat{C}_i| \leq z_i$ and $|\hat{C}_i - E_i| \leq \frac{O(z_i)}{\epsilon}$.

Summing over all i, we get $C_{est}(Y) \geq (1-\epsilon)C_{est} - \sum_i O(z_i)$. Due to the fact that $\sum_i z_i \leq O\left(\epsilon^2 \sqrt{(1+\delta)C_{opt}\hat{C}/\alpha}\right)$, we finally obtain

$$\begin{aligned} &C_{est}(Y) \geq (1-\epsilon)C_{est} - O\left(\epsilon^2 \sqrt{(1+\delta)C_{opt}\hat{C}/\alpha}\right) \\ \Rightarrow &C_{est}(Y) \geq (1 - O(\epsilon))C_{est} \end{aligned}$$

This completes the proof. □

E.6 Proof of Theorem 2

Theorem 1. *For any given query set Q with random arrival order, Algorithm 1 satisfies $\frac{C_{alg}}{C_{opt}} \geq 1 - O(\epsilon + \delta)$ assuming $\frac{C_{opt}}{s_{max}} \geq \Omega(\frac{\alpha M \log(M|\Phi|/\epsilon)}{\epsilon^3(1+\delta)})$, where s_{max} is the maximum performance score obtained for any query, and Φ is an ϵ-net defined over all possible routing strategies $x(\gamma)$.*

Proof. **Case** $\gamma^* \in \Phi$. According to Lemma 4 and 6, we have a union bound over Φ. This union bound implies that with probability greater than $1-\epsilon$, $\sum_i z_i \leq O(\epsilon^2\sqrt{(1+\delta)C_{opt}\hat{C}})$ holds. Therefore, we have that

$$\max\{\hat{C}, F(\gamma^*)\} - C_{est} \leq \frac{1}{\epsilon}\sum_i z_i \Rightarrow \max\{\hat{C}, F(\gamma^*)\} - C_{est} \leq O\left(\epsilon\sqrt{(1+\delta)C_{opt}\hat{C}/\alpha}\right)$$

- When $(1+\delta)C_{opt} \geq \hat{C}/\alpha$, we have $\max\{\hat{C}, F(\gamma^*)\} - C_{est} \leq O(\epsilon(1+\delta)C_{opt})$.

  By weak dualtiy, we have $C_{est} \leq \alpha\hat{C}_{opt} \leq F(\gamma^*)$. Also, we have $\frac{\hat{C}_{opt}}{C_{opt}} \in [1-O(\delta), 1+O(\delta)]$ (from Lemma 3). Therefore, we obtain that $\alpha(1-O(\delta))C_{opt} - C_{est} \leq O(\epsilon(1+\delta)C_{opt}) \Rightarrow C_{est}/\alpha \geq (1-O(\epsilon+\delta))C_{opt}$.

- When $(1+\delta)C_{opt} < \hat{C}/\alpha$, by the same weak dualtiy, we have $\hat{C} - \alpha(1+O(\delta))C_{opt} \leq O(\epsilon\hat{C}/\alpha)$, which further implies that $\hat{C} \leq \alpha\frac{1+O(\delta)}{1-O(\epsilon)/\alpha}C_{opt}$. Therefore, we have $\alpha(1+\delta)C_{opt} - C_{est} \leq O(\epsilon\hat{C}) \Rightarrow C_{est}/\alpha \geq (1-O(\epsilon+\delta))C_{opt}$.

We can define $C_{alg} := \sum_i \min\{T_i, \sum_j d_{ij}x_{ij}(\gamma^*)\}$, where $T_i := \sum_j^t d_{ij}x_{ij}(\gamma^*)$ with $t = \underset{t}{\operatorname{argmax}} \sum_j^t g_{ij}x_{ij}(\gamma^*),\ s.t.\ \sum_j^t g_{ij}x_{ij}(\gamma^*) \leq B_i$.

Furthermore, we have $C_{est}/\alpha = \sum_i \min\{E_i/\alpha, \sum_j \hat{d}_{ij}x_{ij}(\gamma^*)\}$, where $E_i/\alpha = \sum_j^k \hat{d}_{ij}x_{ij}(\gamma^*)$ with $k = \underset{k}{\operatorname{argmax}} \sum_j^k \hat{g}_{ij}x_{ij}(\gamma^*),\ s.t.\ \sum_j^k \hat{g}_{ij}x_{ij}(\gamma^*) \leq B_i$.

Let t_i^* be the largest t such that $\sum_j^{t_i^*} g_{ij}x_{ij}(\gamma^*) \leq B_i$. According to Assumption 1, for each $i \in [M]$, we have

$$(1-O(\delta))\sum_j^{t_i^*}\hat{g}_{ij}x_{ij}(\gamma^*) \leq \sum_j^{t_i^*} g_{ij}x_{ij}(\gamma^*) \leq (1+O(\delta))\sum_j^{t_i^*}\hat{g}_{ij}x_{ij}(\gamma^*)$$

Therefore, we have

$$\sum_j^{t_i^*}\hat{g}_{ij}x_{ij}(\gamma^*) \leq B_i/(1-O(\delta)) = (1+O(\delta))B_i \tag{13}$$

Let k_i^* be the largest k such that $\sum_j^{k_i^*}\hat{g}_{ij}x_{ij}(\gamma^*) \leq B_i$. Without loss of generalizability, assume $t_i^* \geq k_i^*$. Then, there exists a set of tail queries, A_i, which makes Equation (13) exceed B_i by at most $O(\delta)B_i$. Then, we define excess performance contributed by these tail queries as $\sum_{j\in A_i}\hat{d}_{ij}x_{ij}(\gamma^*)$. Because the query order is random, we consider the expected excess performance

$$\mathbb{E}[\sum_{j\in A_i}\hat{d}_{ij}x_{ij}(\gamma^*)] = \frac{O(\delta)B_i}{B_i}E_i/\alpha = O(\delta)E_i/\alpha$$

and the variance satisfies $\textsc{Var}[\sum_{j\in A_i}\hat{d}_{ij}x_{ij}(\gamma^*)] \leq O(\delta)||E_i/\alpha||_2^2$. Using Bernstein's inequality, we have

$$\Pr(|\sum_{j\in A_i}\hat{d}_{ij}x_{ij}(\gamma^*) - O(\delta)E_i/\alpha| > e_i) \leq 2\exp\left(-\frac{e_i^2/2}{O(\delta)||E_i/\alpha||_2^2 + \frac{e_i}{3}s_{max}}\right)$$

Follow Lemma 4, set RHS as $\epsilon/(M|\Phi|)$, use $\frac{C_{opt}}{s_{max}} \geq \Omega(\frac{\alpha M \log(M|\Phi|/\epsilon)}{\epsilon^3(1+\delta)})$, and union bound for all γ^* and i, then with probability greater than $1-\epsilon$,

$$\sum_i e_i \leq O\left(\sqrt{\epsilon^3(1+\delta)O(\delta)C_{opt}E/\alpha^2}\right)$$

where we denote $E = \sum_i E_i$.

Based on this, we have

$$\begin{aligned}
\left|\sum_i T_i - \sum_i E_i/\alpha\right| &\leq \left|\sum_i \sum_j^{t_i^*} d_{ij}x_{ij}(\gamma^*) - \sum_i \sum_j^{k_i^*} \hat{d}_{ij}x_{ij}(\gamma^*)\right| \\
&\leq \left|(1+O(\delta))\sum_i \sum_j^{t_i^*} \hat{d}_{ij}x_{ij}(\gamma^*) - \sum_i \sum_j^{k_i^*} \hat{d}_{ij}x_{ij}(\gamma^*)\right| \\
&\leq \left|O(\delta)\sum_i E_i/\alpha + (1+O(\delta))\sum_{j\in A_i}\sum_i \hat{d}_{ij}x_{ij}(\gamma^*)\right| \\
&\leq \left|O(\delta)\sum_i E_i/\alpha + (1+O(\delta))(O(\delta)E/\alpha + \sum_i e_i)\right| \\
&\leq \left(O(\delta) + O\left(\sqrt{\epsilon^3(1+\delta)O(\delta)C_{opt}/E}\right)\right)\sum_i E_i/\alpha \\
&\leq \left(O(\delta) + O\left(\sqrt{\epsilon^3(1+\delta)O(\delta)/(1-O(\epsilon+\delta)\alpha)}\right)\right)\sum_i E_i/\alpha \\
&\leq \left(O(\delta) + O\left(\epsilon^{3/2}\sqrt{\delta/\alpha}\right)\right)\sum_i E_i/\alpha
\end{aligned}$$

where the penultimate inequality follows $E/\alpha \geq C_{est}/\alpha \geq (1-O(\epsilon+\delta))C_{opt}$.

Finally, it leads to $|C_{alg} - C_{est}/\alpha| \leq \left(O(\delta) + O\left(\epsilon^{3/2}\sqrt{\delta/\alpha}\right)\right)C_{est}/\alpha$. Because $C_{est}/\alpha \geq (1-O(\epsilon+\delta))C_{opt}$, we obtain that $\frac{C_{alg}}{C_{opt}} \geq 1 - O(\epsilon+\delta)$.

Case $\gamma^* \notin \Phi$. As Φ is a ϵ-net, there exists an γ' such that $\forall\, i, j,\ |x_{ij}(\gamma^*) - x_{ij}(\gamma')| \leq \epsilon$. Therefore, we obtain

$$\begin{aligned}
|\hat{C}_i(\gamma^*, P) - \epsilon\hat{C}_i(\gamma^*)| &\leq |\hat{C}_i(\gamma', P) - \epsilon\hat{C}_i(\gamma')| + |\hat{C}_i(\gamma', P) - \hat{C}_i(\gamma^*, P)| + |\epsilon\hat{C}_i(\gamma') - \epsilon\hat{C}_i(\gamma^*)| \\
&\leq z_i' + \epsilon\hat{C}_i(\gamma^*, P) + \epsilon^2\hat{C}_i(\gamma^*) \\
&\leq z_i' + O(\epsilon^2\hat{C}_i(\gamma^*))
\end{aligned}$$

Similarly, for each i, we have

$$\begin{aligned}
|\sum_{j\in A_i} \hat{d}_{ij}x_{ij}(\gamma^*) - O(\delta)E_i(\gamma^*)/\alpha| &\leq |\sum_{j\in A_i} \hat{d}_{ij}x_{ij}(\gamma') - O(\delta)E_i(\gamma')/\alpha| \\
&\quad + |\sum_{j\in A_i} \hat{d}_{ij}x_{ij}(\gamma') - \sum_{j\in A_i} \hat{d}_{ij}x_{ij}(\gamma^*)| \\
&\quad + |O(\delta)E_i(\gamma')/\alpha - O(\delta)E_i(\gamma^*)/\alpha| \\
&\leq e_i' + \epsilon\sum_{j\in A_i} \hat{d}_{ij}x_{ij}(\gamma^*) + \epsilon O(\delta)E_i(\gamma^*)/\alpha \\
&\leq e_i' + O(\epsilon\delta E_i(\gamma^*)/\alpha)
\end{aligned}$$

where, with slight abuse of notation, we use $E_i(\gamma^*)$ to denote the value computed under γ^* and $E_i(\gamma')$ under γ'.

By summing over all i, and following Lemma 4 as well as the proof in case $\gamma^* \in \Phi$, we obtain $C_{est}/\alpha \geq (1 - O(\epsilon + \delta))C_{opt}$, and $|C_{alg} - C_{est}/\alpha| \leq \left(O(\delta) + O\left(\epsilon^{3/2}\sqrt{\delta/\alpha}\right)\right) C_{est}/\alpha$.

Therefore, Algorithm 1 satisfies that $\frac{C_{alg}}{C_{opt}} \geq 1 - O(\epsilon + \delta)$. $\square$