FLARE: Fine-Grained Length-Aware Routing for Resource-Efficient Heterogeneous LLM Serving

Yujia Fu[1†], **Heming Zhong**[1†], **Dan Huang**[1*], **Yutong Lu**[1*]
[1]School of Computer Science and Engineering, Sun Yat-sen University
{fuyj25@mail2, zhonghm5@mail2, huangd79@mail, luyutong@mail}.sysu.edu.cn

Abstract

With the rapid proliferation of large language models (LLMs), model pools have become increasingly heterogeneous in both capability and efficiency. Larger LLMs can improve quality but incur higher latency and cost, while smaller LLMs are the opposite, making per-query model selection crucial in practice. This has spawned LLM routers that dispatch each query to an appropriate model. Existing routers lack fine-grained resource awareness across deployment settings, which degrades efficiency metrics in real-world serving. To this end, we propose FLARE, a length-centric, resource-aware multi-LLM routing framework that uses length-based models to estimate per-query latency and cost. FLARE formulates routing as a discrete multi-objective optimization problem to achieve an efficient trade-off. Experiments show that FLARE reduces latency and cost by up to 68% and 75% while achieving sufficient accuracy, and can be easily applied to new datasets and LLMs.

1 Introduction

Large Language Models (LLMs) have achieved strong performance on a broad spectrum of challenging, domain-specific tasks (Liu et al., 2024; Guo et al., 2025a; Xue et al., 2024; OpenAI, 2024). Major LLM providers have introduced model families spanning a wide range of parameter scales, such as the Qwen and Llama series (Bai et al., 2023; Grattafiori et al., 2024). These models exhibit heterogeneous behaviors (Figure 1) not only in accuracy, but also in latency and monetary cost, with performance varying across tasks, parameter sizes, deployment environments, and query concurrency. In practice, such heterogeneity brings out distinct service-level objectives (Huang et al., 2025; Li et al., 2025) across application scenarios. For instance, an e-commerce assistant (Sun et al., 2022) can rely on a fine-tuned 3B model deployed to meet strict tail-latency constraints (Xu et al., 2024). In contrast, coding assistants requiring multi-step reasoning over long documents (Guo et al., 2025b; Sun et al., 2025) can tolerate higher latency and cost (Chen et al., 2023). Therefore, to achieve an optimal trade-off among the objectives of accuracy, latency, and cost, routing queries across heterogeneous models is a promising solution.

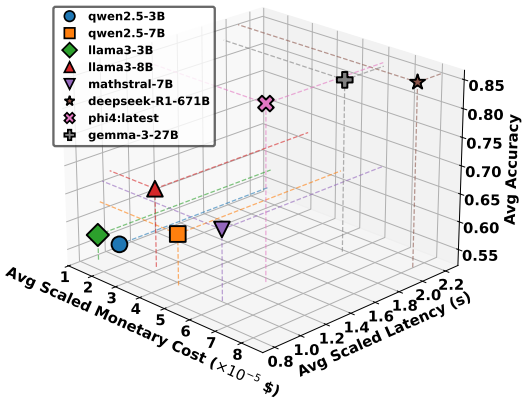

Figure 1: Each point shows an LLM in a 3D space of average accuracy (z), normalized scaled latency (y), and normalized scaled monetary cost (x).

Building on this observation, recent model-routing methods address this challenge by training custom routers that dispatch each query to a selected LLM, aiming to improve response quality or reduce serving costs under coarse budget constraints. However, existing approaches primarily optimize router accuracy while ignoring efficiency metrics. Some works abstract away the resource consumption and focus on improving accuracy performance (Ding et al., 2024; Ong et al., 2025). Other routing methods model latency and cost with coarse approximation that does not generalize across workloads or infrastructures (Stripelis et al., 2024; Song et al., 2025; Hu et al., 2024; Yu

† Equal contribution.
* Corresponding authors.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 22249–22266

et al., 2025). As a result, without fine-grained latency modeling, routers cannot adapt to varying of query sizes and types at runtime.

These limitations underscore the need for an LLM routing system that can achieve an efficient trade-off between model accuracy and system-level performance in real-world deployments.

To this end, we propose FLARE, a **Fine-Grained Length-Aware routing framework for Resource-Efficient heterogeneous LLM serving**. Based on the observed proportional relationship between output length and key efficiency metrics, FLARE explicitly estimates per-query latency and cost. This metric awareness allows FLARE to formulate routing as a multi-objective optimization problem. FLARE further provides scenario-adaptive configurations, enabling it to dynamically satisfy user-specified trade-offs such as efficiency-first. FLARE estimates objective metrics by extracting and learning rich query- and model-level features. This estimation method can be migrated to out-of-distribution datasets and applied to newly introduced LLMs via limited warm-up queries. Our contributions are as follows:

- We present a length-aware, fine-grained model of LLM efficiency metrics, enabling explicit per-query performance characterization across heterogeneous LLMs.
- We propose a novel resource-aware multi-objective routing policy that selects an appropriate LLM to satisfy diverse serving objectives, ensuring flexible performance-cost trade-offs.
- Through experiments on both in-distribution and out-of-distribution workloads, FLARE consistently outperforms baselines across key metrics. Reducing mean latency by up to 68% and cost by up to 75% while maintaining competitive accuracy; experiments further validate the effectiveness of cold-start adaptation for newly introduced LLMs.

2 Related Work

Cost-efficient LLM routing. LLM routing selects an appropriate model for each query without invoking all candidates. Prior work mainly follows two paradigms. Non-predictive methods obtain partial signals from models or retrieval to estimate whether generation is necessary, such as cascaded invocation under a quality threshold (Chen et al., 2023; Narayan et al., 2025; Chuang et al., 2024) or retrieval-first prechecks that bypass costly inference (Zhao et al., 2024; Patidar et al., 2025). Predictive methods train a lightweight router to directly choose an LLM based on query features. These methods include binary routing between an LLM pair (Ding et al., 2024; Ong et al., 2025) and multi-expert selection trained with soft quality labels (Stripelis et al., 2024; Song et al., 2025; Ding et al., 2025). These routers often treat each LLM's efficiency as a coarse constraint (Stripelis et al., 2024; Ong et al., 2025; Hu et al., 2024; Wang et al., 2025a; Fernandez et al., 2025).

These existing routers remain largely accuracy-driven, with limited awareness of explicit efficiency metrics. This limitation becomes more severe in heterogeneous multi-LLM deployments. In this paper, we try to bridge this gap by integrating per-query latency and cost awareness, enabling cost-efficient decisions across LLMs and workloads.

LLM efficiency metric prediction. To characterize inference budgets in LLM serving, prior works usually use output length as a core signal to estimate system-level metrics like end-to-end latency and monetary cost (Han et al., 2025; Fu et al., 2024; Wang et al., 2025b). S^3 (Jin et al., 2023) predicts output length with lightweight BERT models to guide packing of inference requests. The method by Zheng et al. (Zheng et al., 2023) trains a length-perception head and uses the predicted length to estimate generation latency and to form length-homogeneous micro-batches. μ-Serve (Qiu et al., 2024) further combines length categorization with serving-layer co-optimization of power and latency.

These methods prove that length prediction is effective during practical LLM serving. Inspired by these methods, we extend length prediction into routing for heterogeneous LLM pools and diverse workloads.

3 Preliminaries

To elucidate the insight of interplay among accuracy, latency, and cost, we conducted two comprehensive studies using typical LLM inference workloads. The observations provide empirical motivation for our resource-aware trade-off design.

3.1 Observation 1: Latency and Cost Dominate Once Accuracy Is Thresholded

Across diverse LLM inference workloads, we observe a pronounced accuracy-efficiency trade-off, where latency and cost still vary substantially among models that answer a query correctly. To isolate this effect, we analyze model-query pairs

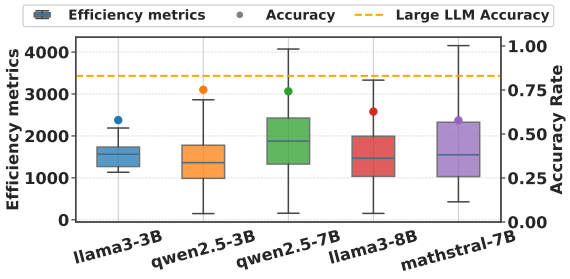

Figure 2: Distribution of efficiency metrics over per-query candidates that pass accuracy threshold in MMLU, cost and latency is aggregated into a unified metric

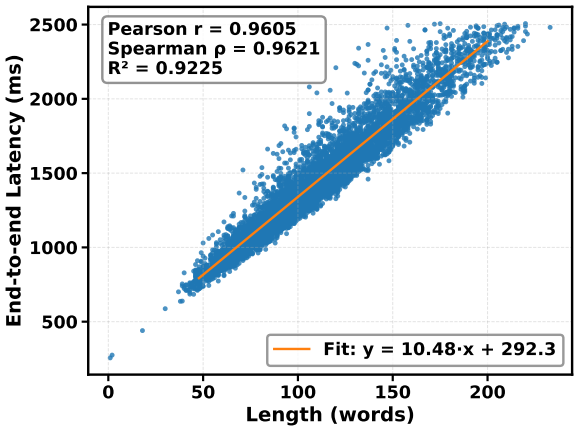

Figure 3: DeepSeek-R1-671B's end-to-end latency versus output length with a linear fit

with estimated accuracy above the threshold in the MMLU dataset (Hendrycks et al., 2020). As shown in Figure 2, within this accuracy-feasible set, joint efficiency metrics show substantially larger dispersion than accuracy. This asymmetry suggests that optimizing latency and cost among accuracy-qualified candidates is feasible to achieve a better trade-off.

3.2 Observation 2: Output Length Strongly Predicts Latency and Cost

To support fine-grained resource-aware routing, we require per-query estimates of latency and serving cost. Modeling all efficiency factors directly is impractical, but we confirm that output length is a strong predictor for both latency and cost. We profile each candidate LLM on full datasets, collecting the generated length, end-to-end latency, and monetary cost. Results in Figure 3 reveal that latency scales approximately linearly with output length. Output length also correlates strongly with monetary cost since most APIs bill by generated tokens. This observation motivates our per-query latency and cost estimation, which makes efficiency explicit and enables routing under different deployment scenarios.

4 Design

4.1 Problem Definition

Assume $\mathcal{X}$ denotes the query sets and $\mathcal{M}$ the LLM pool. We partition $\mathcal{M}$ into a set of small language models $\mathcal{M}_{\mathrm{S}}$, which deliver lower-quality responses but at a lower cost, and a set of large language models $\mathcal{M}_{\mathrm{L}}$, which prioritize response quality at a higher cost.

For each query $x \in \mathcal{X}$ and LLM $m \in \mathcal{M}$, we consider the following variables:

- $A_m(x) \in \{0, 1\}$, if m answers x correctly;
- $L_m(x)$, the number of generated words;
- $T_m(x)$, the end-to-end latency;
- $C_m(x)$, the monetary cost.

Then we define a router as a function $R\colon \mathcal{X} \to \mathcal{M}$ that selects one LLM per query. For each candidate m, we associate the objectives $\big(\Pr[A_m(x) = 1],\, T_m(x),\, C_m(x)\big)$. We treat routing as a discrete multi-objective optimization problem over LLMs.

Given an accuracy threshold $\tau \in (0, 1)$, we define the accuracy-qualified set

$$\mathcal{M}_\tau(x) = \{m \in \mathcal{M} \mid \Pr[A_m(x) = 1] \geq \tau\}. \quad (1)$$

Within $\mathcal{M}\tau(x)$, we define domination over feasible candidates: a model m dominates m' if it is no worse in both latency and cost and strictly better in at least one. The per-query Pareto frontier $\mathcal{F}_\tau(x)$ is then the set of all non-dominated models, representing the ideal routing trade-off surface in the (T, C) space under the accuracy constraint.

$$\begin{aligned}\mathcal{F}_\tau = \Big\{ m \in \mathcal{M}_\tau \;\Big|\; \nexists\, m' \in \mathcal{M}_\tau \text{ s.t.} \\ (T_{m'}, C_{m'}) \leq (T_m, C_m), \\ (T_{m'}, C_{m'}) \neq (T_m, C_m) \Big\}.\end{aligned} \quad (2)$$

To obtain a single operating point, we use a weighted objective P_m, which effectively induces a per-query model selection rule over $\mathcal{M}$.

$$P_m(x) = \alpha\, T_m(x) + \beta\, C_m(x). \quad (3)$$

where (α, β) reflects the latency-cost preference of the target scenario. We can also add an optional accuracy weight ω to incorporate accuracy into the objective. The ideal routing decision is in:

$$m^*(x) = \arg \min_{m \in \mathcal{M}_\tau(x)} P_m(x). \quad (4)$$

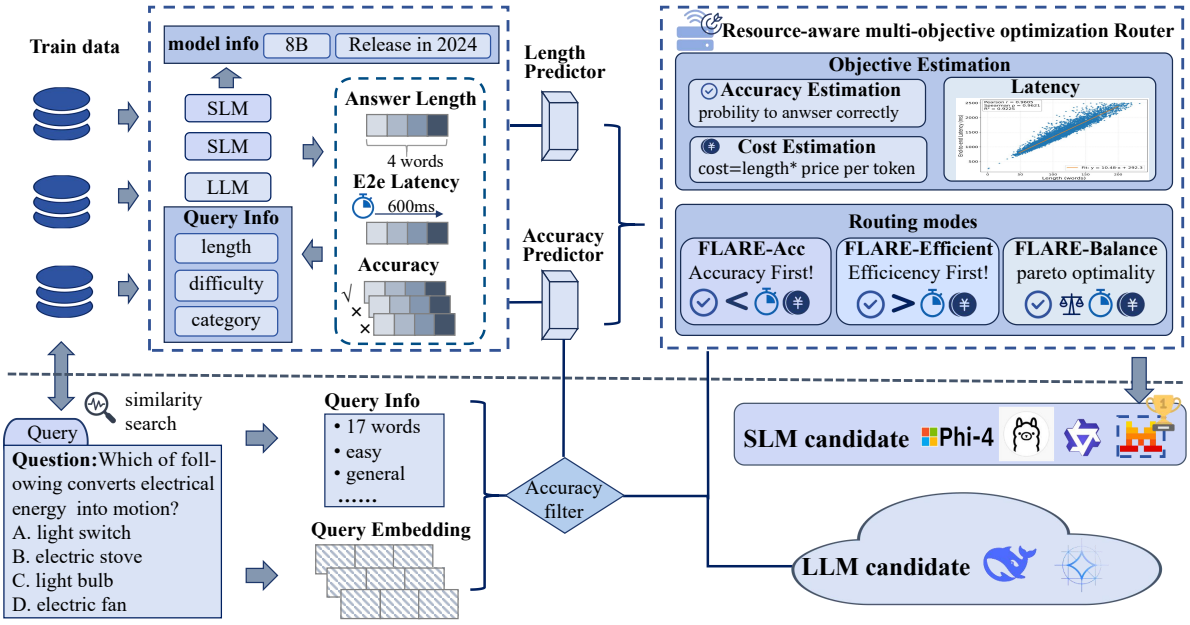

Figure 4: FLARE overview architecture.

To quantify how close a selected LLM is to the ideal set, we use the inverted generational distance (IGD) (Reyes Sierra and Coello Coello, 2005), which measures the average distance from the Pareto-optimal reference set to the chosen LLM. We compute it over all queries and weight each query by the size of its ideal set as follows.

$$\mathrm{IGD}_{\mathrm{overall}} = \frac{1}{\sum_{\tau\in\mathcal{D}} |\mathcal{F}_\tau|} \sum_{\tau\in\mathcal{D}} \sum_{m_f\in\mathcal{F}_\tau} \left\| (\tilde{T}_{m_f}, \tilde{C}_{m_f}) - (\tilde{T}_{m_\tau}, \tilde{C}_{m_\tau}) \right\|_2. \tag{5}$$

where $(\tilde{T}, \tilde{C})$ are normalized versions of (T, C) to account for scale differences. Meanwhile, the formulation is tunable, where τ sets the minimum acceptable accuracy and (α, β) controls the trade-off between latency and cost.

4.2 System Overview

Figure 4 shows FLARE's overall design, where we separate two main stages, offline profiling and online serving. In the offline profiling stage, FLARE measures variables $(A_m(x), L_m(x), T_m(x), C_m(x))$, fitting predictors for accuracy and length. Then latency and cost models are derived from the predicted length. In the online serving stage, FLARE adapts its router to each incoming query by filtering candidates based on its accuracy constraint and optimizing a configurable latency-cost objective using predicted per-candidate metrics, thereby providing a practical approximation to Eq. (4).

4.3 Offline Profiling and Predictor Training

Data profiling and feature extraction. For each query x and LLM m, we collect all metrics from the LLM output, including generated length $L_m(x)$, end-to-end latency $T_m(x)$, and correctness $A_m(x)$.

At this stage, FLARE's goal is to extract features that can guide online serving. For each query, FLARE encodes a semantic embedding $e(x) \in \mathbb{R}^{384}$ using all-MiniLM-L6-v2. Additionally, lightweight query attributes are extracted, including prompt lengths and dataset types. As for query difficulty, we compute a cross-LLM correctness rate $r(x) = \frac{1}{|\mathcal{M}|} \sum_{m\in\mathcal{M}} A_m(x)$, and bucket it into three levels.

For each LLM, we collect descriptors such as parameter size, release year, coarse knowledge coverage, and serving mode.

Assume $q(x)$ denotes the explicit query attributes and $g(m)$ denotes the LLM descriptors. We form a joint feature vector in Eq. (6).

$$\phi(x, m) = \big[e(x),\ q(x),\ g(m)\big], \tag{6}$$

which serves as inputs to the accuracy predictor and output length predictor.

Predictor training for accuracy and output length Given $\phi(x, m)$, we train both the accuracy predictor and the output length predictor across all LLMs and datasets. The accuracy predictor f_{acc} first outputs a correctness probability $\hat{p}_m(x) \in [0, 1]$,

$$\hat{p}_m(x) = f_{\text{acc}}(\phi(x, m)), \tag{7}$$

and we mark the response as correct if $\hat{p}_m(x) \geq \tau$,

$$\hat{A}_m(x) = \mathbb{I}[\hat{p}_m(x) \geq \tau] \tag{8}$$

The output length predictor f_{len} maps $\phi(x, m)$ to a predicted output length,

$$\hat{L}_m(x) = f_{\text{len}}(\phi(x, m)). \tag{9}$$

Both predictors are deployed as lightweight XGBoost models. We also compare XGBoost with a BERT-style predictor and find that XGBoost is better suited to our tabular feature design. Implementation details and training results are further reported in Appendix B.

Length-guided latency and cost modeling. Since latency scales with output length in a model-dependent manner, we fit a per-LLM linear latency model:

$$\hat{T}_m(x) = k_m \, \hat{L}_m(x) + b_m, \tag{10}$$

where (k_m, b_m) are learned from offline profiling by regressing the measured latency $T_m(x)$ on the real output length $L_m(x)$. Overall, predicting latency from predicted output length yields $R^2 = 0.616$ on the full dataset, which indicates that the predicted output length provides a meaningful signal for latency modeling.

Since fitting (k_m, b_m) requires a set of latency-length samples, FLARE introduces an extra online policy for a newly introduced LLM. When a new LLM is initialized, FLARE bootstraps (k_m, b_m) from a few early latency-length traces and then refines the regression during serving. Hence, the latency regression quickly converges with limited data. This cold-start strategy allows FLARE to expand model pools with low overhead.

As for the monetary cost, we compute it with length-based pricing:

$$\hat{C}_m(x) = \pi_m^{\text{in}} \, \ell_{\text{in}}(x) + \pi_m^{\text{out}} \, \hat{L}_m(x), \tag{11}$$

where $\ell_{\text{in}}(x)$ is the input length and $(\pi_m^{\text{in}}, \pi_m^{\text{out}})$ are LLM-specific prices.

We further discuss the impact of output-length prediction error on FLARE's performance in Appendix F, which supports the reliability of FLARE's current length-prediction component.

4.4 Online Resource-aware Routing

Query characterization. For an incoming query x, FLARE embeds it with an all-MiniLM-L6-v2[1] encoder and retrieves its k nearest profiled queries via KNN. For unseen datasets, FLARE infers dataset types and difficulties from these neighbors by aggregating their metadata and cross-LLM correctness statistics. Together with prompt length, the inferred attributes form the query-side features in $\phi(x, m)$. Details for KNN feature extractors are in Appendix B.

Accuracy-based filtering. FLARE applies f_{acc} to enforce the constraint $\hat{A}_m(x) \geq \tau$ in Eq. (1). FLARE prefers feasible small LLMs $m_s \in \mathcal{M}_{\text{S}}$ and falls back to large LLMs $m_\ell \in \mathcal{M}_{\text{L}}$ only when no m_s is qualified. If no m meets the constraint, FLARE relaxes τ and chooses the smallest $\hat{P}_m(x)$.

Scenario-specific LLM routing. After estimating accuracy and efficiency metrics, FLARE offers three routing modes to match different accuracy-efficiency objectives.We provide a sensitivity analysis of τ, α, β, and ω in Appendix E.

(1) FLARE-ACC. For accuracy-critical settings, FLARE applies a stricter accuracy filter and selects the target LLM with Eq. (12).

$$m^\star = \arg\max_m \; \hat{p}_m(x). \tag{12}$$

(2) FLARE-EFFICIENT. When efficiency is the priority under a basic accuracy requirement, FLARE restricts to $\hat{p}_m(x) \geq \tau$ and minimizes the resource objective in Eq. (3). The weights (α, β) are tuned on a validation set using IGD (Eq. (5)) to obtain operating points near the Pareto frontier.

(3) FLARE-BALANCE. For users without a clear preference, we provide a balanced setting that minimizes a joint objective over efficiency and accuracy, assuming $\hat{e}_m(x) = 1 - \hat{p}_m(x)$ denotes the predicted error probability:

$$\hat{P}_m(x) = \alpha \, \hat{T}_m(x) + \beta \, \hat{C}_m(x) + \omega \, \hat{e}_m(x), \tag{13}$$

where we select the target LLM $m^\star = \arg\min_m \hat{P}_m(x)$. We tune (α, β, ω) on the validation set via IGD to recommend a near-frontier operating point.

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

5 Experiments

5.1 Experimental Setup

We evaluate FLARE on five in-distribution (ID) benchmarks that span common LLM use cases. GSM8K focuses on math (Cobbe et al., 2021), ARC-Easy focuses on science QA (Clark et al., 2018), MBPP evaluates programming (Austin et al., 2021), MMLU tests multi-domain knowledge (Hendrycks et al., 2020), and MUSR targets multi-step reasoning (Sprague et al., 2023). To assess robustness under distribution shift, we additionally use two out-of-distribution (OOD) datasets. Commonsense_QA probes commonsense reasoning (Talmor et al., 2019), and MATH covers competition-level mathematics (Hendrycks et al., 2021).

For each ID dataset, we use an 8:1:1 train/val/test split and then combine the splits across datasets. We train FLARE on the ID training set and tune τ and Eq. (3) on the ID validation set. OOD datasets are used only for testing.

Models. To emulate heterogeneous deployments, we group models into three tiers. The cloud-LLM tier includes API-accessed models (e.g., DeepSeek-R1-671B, Gemma3-27B) with high accuracy but higher latency and cost, which represents the cloud-based closed-source LLM. On the edge, we define two small language model (SLM) tiers to reflect practical on-device memory budgets. Capable SLMs run on an Apple M4 platform (e.g., Qwen2.5-7B-Instruct, Mathstral-7B, Llama3-8B, Phi-4:latest). Cost-efficient SLMs run on a consumer RTX 3060 GPU (e.g., Qwen2.5-3B-Instruct, Llama3.2-3B, DeepSeek-R1-1.5B, Phi-4-Mini). This tiered setup preserves hardware-driven latency differences that would be obscured on a single high-end device. All locally served models use matched settings (see Appendix A).

Baselines. To our knowledge, FLARE is the first multi-LLM router to optimize per-query latency and monetary cost as first-class objectives. We therefore compare against prior routers and FLARE-style heuristics that share the same accuracy filtering, isolating the effect of resource-aware selection.

(1) State-of-the-art routers. TensorOpera (Stripelis et al., 2024) is quality-driven, using BERT-score-style supervision to route for correctness without modeling per-query latency or cost. RouterBench (Hu et al., 2024) is a coarse cost-aware baseline that selects a fixed per-task model set from an accuracy-cost Pareto frontier, ignoring latency and per-query cost variation.

(2) FLARE-derived baselines. These methods share the same accuracy-threshold filter with FLARE and differ only in feasible-set selection. Cand-Random samples uniformly; Acc-Best picks the feasible model with the highest $\hat{p}_m(x)$. Oracle uses ground-truth metrics and minimizes Eq. (3) with Pareto-tuned weights, serving as an ideal strategy.

Metrics. We evaluate routers along task accuracy, efficiency metrics, and routing behavior.

For accuracy, we use benchmark-standard metrics. We report exact match or pass@1 on GSM8K, MATH, and MBPP, and multiple-choice accuracy on MMLU, ARC-Easy, and Commonsense_QA.

For efficiency metrics, we report mean end-to-end latency and mean per-query monetary cost. Cost is computed from length-based pricing using input/output rates collected from OpenRouter[2] or provider websites. (see Appendix A)

For routing behavior, we report the fraction of queries routed to each model and compare it with an Oracle allocation. We also report IGD (Eq. (5)) to measure how close the router's choices are to the per-query Pareto frontier in the latency-cost space.

5.2 Experimental Results

Overall performance on ID and OOD workloads. Table 1 and 2 show that FLARE achieves a better accuracy-efficiency trade-off than prior routers on both ID and OOD workloads. On ID datasets, compared with Acc-Best, FLARE-Acc preserves similar accuracy and reduces latency and cost by up to 59% and 69% respectively. FLARE-Efficient targets resource minimization. Compared with RouterBench and TensorOpera, it reduces latency by 26%-68% and cost by 44-75%. These gains require only a modest accuracy sacrifice of 5%-20% across datasets, which is acceptable for efficiency-oriented deployments. FLARE-Balance provides a balanced trade-off and yields consistently smaller IGD, indicating selections closer to the per-query Pareto frontier in the (T, C) space. In contrast, TensorOpera is accuracy-centric and tends to incur higher resource usage, while RouterBench fixes a small per-dataset model subset on an offline accuracy-cost frontier; it particularly shows

[2] https://openrouter.ai/models

Table 1: Overall routing performance across five ID datasets. Best and second-best results are computed among non-oracle methods: Best per column is bold; second-best is underlined. The ↑ symbol indicates that higher values are better, while ↓ indicates that lower values are better. Cost is reported in units of 10^{-4} USD, and latency is measured in seconds.

Settings	ARC-Easy				GSM8K				MBPP				MMLU				MuSR			
	Acc ↑	Lat ↓	Cost ↓	IGD ↓	Acc ↑	Lat ↓	Cost ↓	IGD ↓	Acc ↑	Lat ↓	Cost ↓	IGD ↓	Acc ↑	Lat ↓	Cost ↓	IGD ↓	Acc ↑	Lat ↓	Cost ↓	IGD ↓
Tensor Opera.	0.936	5.54	0.677	0.206	0.948	6.78	0.306	0.199	0.422	4.25	0.596	0.146	0.835	15.22	8.60	0.371	0.604	22.54	14.86	0.396
RouterBench	0.843	4.77	0.165	0.187	0.876	4.73	0.097	0.123	0.405	2.25	0.063	0.072	0.684	8.61	**0.411**	0.251	0.415	11.24	**0.502**	0.199
Acc-Best	0.962	6.95	0.902	0.268	0.950	8.43	0.366	0.256	**0.433**	5.76	0.950	0.219	0.862	17.80	10.23	0.455	0.630	24.32	16.05	0.471
Cand-Random	0.891	6.57	2.43	0.289	0.858	7.43	2.41	0.229	0.388	5.46	1.66	0.183	0.778	13.98	6.59	0.378	0.600	19.54	12.34	0.400
FLARE-Acc	**0.981**	3.83	0.373	0.140	**0.960**	3.46	0.309	0.082	0.421	0.80	0.037	**0.017**	**0.885**	7.93	3.18	0.155	**0.680**	12.98	7.47	0.191
FLARE-Efficient	0.883	**1.54**	**0.092**	0.045	0.776	**2.27**	**0.058**	0.048	0.353	**0.78**	**0.016**	0.038	0.732	**6.23**	2.60	0.117	0.542	**10.04**	5.07	**0.164**
FLARE-Balance	0.947	2.02	0.217	**0.029**	0.922	2.86	0.154	**0.032**	0.401	1.13	0.031	0.029	0.820	6.39	2.60	**0.103**	0.603	10.43	5.67	**0.164**
Oracle	0.989	1.57	0.115	0.020	0.990	1.51	0.076	0.008	0.533	0.872	0.16	0.003	0.970	3.27	0.773	0.016	0.921	4.32	0.839	0.028

Table 2: Routing performance on OOD datasets.

Settings	commonsense_qa				math			
	Acc ↑	Lat ↓	Cost ↓	IGD ↓	Acc ↑	Lat ↓	Cost ↓	IGD ↓
TensorOpera.	0.875	10.67	4.05	0.260	0.883	8.55	4.39	0.263
RouterBench	0.750	10.52	**0.369**	0.238	0.490	4.05	**0.097**	0.127
Acc-Best	0.880	11.39	3.69	0.305	0.890	9.75	4.95	0.314
Cand-Random	0.737	10.57	5.11	0.316	0.801	7.91	4.404	0.300
FLARE-Acc	**0.903**	6.16	0.739	0.131	**0.891**	4.67	0.782	0.123
FLARE-Efficient	0.761	**4.26**	0.500	0.081	0.795	**2.57**	0.495	0.081
FLARE-Balance	0.855	5.40	0.571	**0.062**	0.861	3.11	0.547	**0.061**
Oracle	0.968	3.44	0.307	0.031	0.977	2.37	0.344	0.027

the lowest cost, but often comes from over-routing to a few cheap small models.

Moreover, FLARE shows consistent trends under OOD shift. FLARE's three modes remain consistent under OOD shift, spanning accuracy-first, efficiency-first, and balanced Pareto-optimal routing with the lowest IGD. RouterBench remains cost-efficient in OOD, but its fixed per-dataset selection yields a sharp accuracy drop.

Routing behavior analysis. Figure 5 illustrates how routing strategies allocate queries. Accuracy-driven baselines concentrate traffic on the most capable cloud models (e.g., Gemma3-27B, DeepSeek-R1-671B) which raises latency and cost. RouterBench routes within a small fixed subset of low-cost models, producing rigid allocations and under-serving hard queries. FLARE adapts choices per request. FLARE-Balance yields a distribution closest to Oracle by emphasizing the same Pareto-relevant models,aligns with its low IGD. FLARE-Acc shifts massively toward capable edge models (e.g., Phi-4, Llama3-8B) to protect accuracy, whereas FLARE-Efficient favors smaller models (e.g., Qwen2.5-3B, DeepSeek-R1-1.5B) to minimize resource usage. We further conduct a qualitative case study of router decision in Appendix D.

5.3 Ablation Results

To quantify the contribution of FLARE's key components, we ablate three mechanisms and report results averaged over datasets.

(1) Pareto selection. This component is used by FLARE-BALANCE and FLARE-EFFICIENT. FLARE-ACC does not use it because it optimizes accuracy directly. We ablate Pareto selection by setting $\alpha = 0$ or $\beta = 0$ in Eqs. (3) and (13) to collapse routing into a single-efficiency objective.

(2) Accuracy-threshold filtering. This component is utilized by FLARE-ACC and FLARE-EFFICIENT. FLARE-BALANCE does not use it because accuracy is already incorporated in its objective and parameter tuning. We ablate filtering by removing the threshold constraint.

(3) Per-query length awareness. This component is utilized by FLARE-BALANCE and FLARE-EFFICIENT, related to length-based efficiency prediction. We replace $\hat{L}_m(x)$ with a model-level mean length to assess the impact of removing per-query length signals on latency and cost estimation.

Figure 6 shows consistent trends. Removing Pareto selection yields unbalanced latency–cost trade-offs. Removing the threshold increases latency and cost for FLARE-ACC and causes large accuracy drops for FLARE-EFFICIENT. Replacing per-query length with mean length preserves accuracy but worsens latency and cost, indicating that instance-level length prediction is critical.

5.4 Cold Start for LLM Scalability

To evaluate the effectiveness of our cold-start strategy for new models, we perform a leave-one-

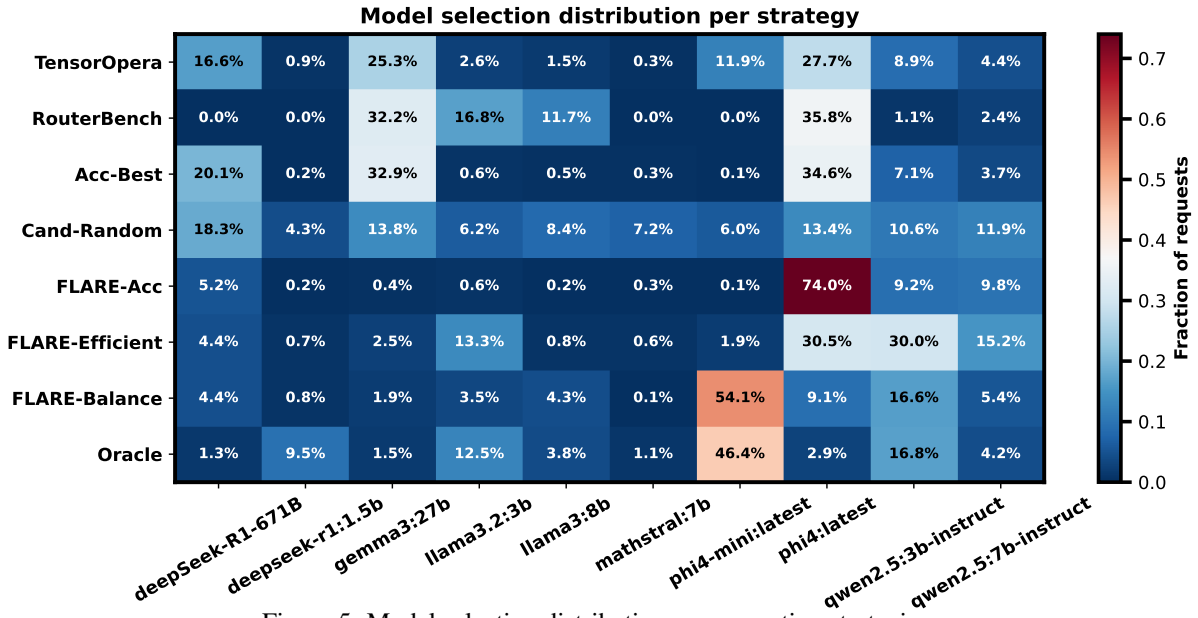

Figure 5: Model selection distribution across routing strategies.

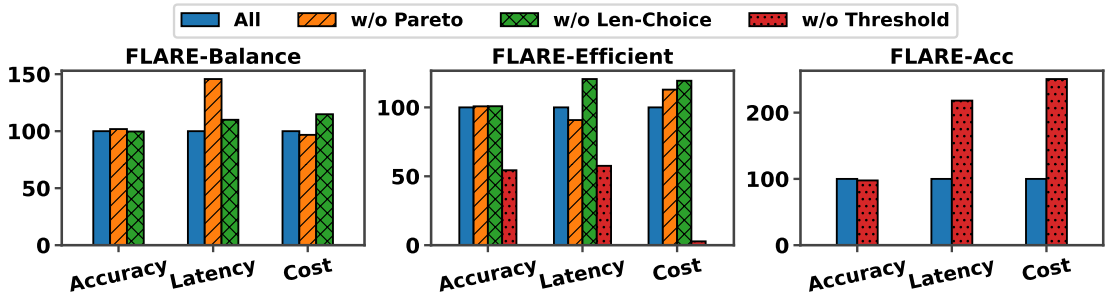

Figure 6: Ablation results averaged over all datasets. Rows correspond to ablated components (w/o Pareto, w/o Threshold, w/o Len-Choice), and columns report metrics (Accuracy, Latency, Cost). The x-axis shows FLARE variants involved in each ablation. Cost values are reported in units of 10^{-2}, and latency is reported in second.

model-out study. We treat each LLM in turn as newly introduced and use remaining LLMs as in-system pool. For target LLM, we split its data into individual train and test set. We process train set in batches of 25 requests in step and incrementally update FLARE's accuracy and length predictors. We then evaluate the predictors on the test set after each batch. Figure 7 reports length-based latency MAE and accuracy F1 over steps for Llama3-8b (Others in Appendix C). We observe the predictors typically reach near in-system performance after about 200 requests, indicating that FLARE adapts quickly to unseen models with limited data. As for retraining overhead of XGBoost, online adaptation is lightweight. XGBoost updates run on CPU and can be overlapped with GPU inference.

5.5 Routing overhead of FLARE.

We measure the runtime overhead of FLARE's router, including feature preparation, XGBoost inference, and the routing decision. Across three runs of test queries, the average routing overhead is 18.12 ± 0.02 milliseconds per query, compared to 1500 milliseconds mean end-to-end latency of the fastest model Qwen2.5:3B in our pool; thus, the routing overhead is negligible in our setting.

6 Conclusion

We presented FLARE, a fine-grained length-aware and resource-aware router for heterogeneous multi-LLM systems. By combining length-based efficiency modeling with accuracy-thresholded selection, FLARE turns routing into a constrained resource optimization problem at per-request granularity. Experiments show that FLARE reduces mean latency by up to 68% and cost by up to 75% while achieving sufficient accuracy, and it generalizes well to out-of-distribution workloads and newly introduced LLMs.

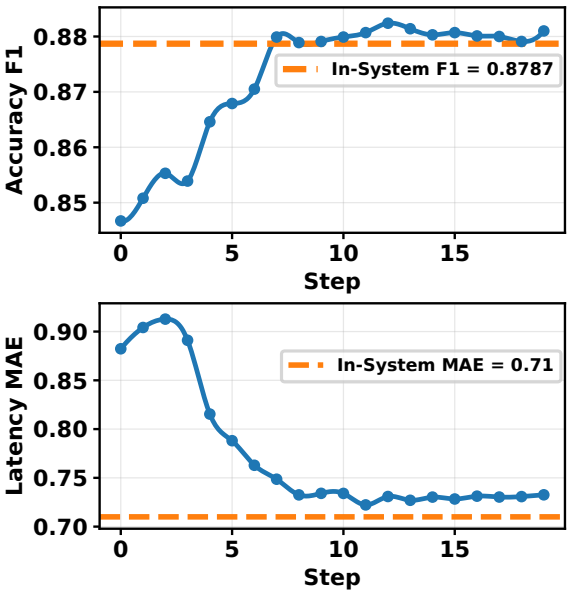

Figure 7: cold start performance in accuracy prediction and latency modeling over steps.

Limitations

Length and latency modeling. Despite our efforts to control deployment conditions, output-length and latency prediction remains a complex problem influenced by many external factors beyond the router's control, such as runtime contention and system-level variability; therefore, there is still substantial room to further improve prediction accuracy. Future work could incorporate hardware-level signals and serving mechanisms such as KV-cache dynamics and early stopping to strengthen the length–latency mapping, and potentially extend FLARE to multi-turn or per-token routing.

Accuracy modeling. We model accuracy as a binary outcome and do not leverage response content to characterize finer-grained answer quality. In Future work, FLARE's resource-aware formulation can be combined with richer quality estimators to jointly improve sensitivity to accuracy, latency, and cost.

Workload coverage. Our evaluation uses standard labeled benchmarks whose queries are relatively short and cannot fully represent the diversity of real-world traffic. Expanding to broader, more realistic workloads is an important next step for validating FLARE in production settings.

Configuration granularity. While FLARE provides three preset operating modes that cover many practical needs, users cannot directly tune parameters such as (α, β) to perform a finer-grained and more complete scan of the cost-performance trade-off surface. This suggests that more systematic exploration and calibration of latency-cost objectives and their weighting is needed to better support concise deployment preferences.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback and suggestions. This research was supported by the National Key R&D Program of China (2025YFB3003701, 2025YFB3003700). This work was supported in part by the Guangxi Key Research and Development Program under Grant GuikeAB25069495.

References

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

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and 1 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Lingjiao Chen, Matei Zaharia, and James Zou. 2023. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *arXiv preprint*. ArXiv:2305.05176 [cs].

Yu-Neng Chuang, Prathusha Kameswara Sarma, Parikshit Gopalan, John Boccio, Sara Bolouki, Xia Hu, and Helen Zhou. 2024. Learning to route llms with confidence tokens. *arXiv preprint arXiv:2410.13284*.

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

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

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

Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian Del Carmen Hipolito Garcia, Menglin Xia, Laks VS Lakshmanan, Qingyun Wu, and Victor Rühle. 2025. Best-route: Adaptive llm routing with test-time optimal compute. *arXiv preprint arXiv:2506.22716*.

Nigel Fernandez, Branislav Kveton, Ryan A Rossi, Andrew S Lan, and Zichao Wang. 2025. Radar: Reasoning-ability and difficulty-aware routing for reasoning llms. *arXiv preprint arXiv:2509.25426*.

Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. 2024. Efficient llm scheduling by learning to rank. *Advances in Neural Information Processing Systems*, 37:59006–59029.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025a. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Jinyao Guo, Chengpeng Wang, Xiangzhe Xu, Zian Su, and Xiangyu Zhang. 2025b. Repoaudit: An autonomous llm-agent for repository-level code auditing. *arXiv preprint arXiv:2501.18160*.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. Token-budget-aware llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24842–24855.

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

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

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. 2024. Routerbench: A benchmark for multi-llm routing system. In *ICML 2024 Workshop on Agentic Markets*. Poster.

Jinqi Huang, Yi Xiong, Xuebing Yu, Wenjie Huang, Entong Li, Li Zeng, and Xin Chen. 2025. Slo-aware scheduling for large language model inferences. *arXiv preprint arXiv:2504.14966*.

Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. 2023. S^3: Increasing GPU utilization during generative inference for higher throughput. In *Advances in Neural Information Processing Systems*, volume 36, pages 18015–18027.

Mingjia Li, Hong Qian, Jinglan Lv, Mengliang He, Wei Zhang, and Aimin Zhou. 2025. Foundation model enhanced derivative-free cognitive diagnosis. *Frontiers of Computer Science*, 19(1):191318.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Avanika Narayan, Sabri Eyuboglu, Dan Biderman, Avner May, Scott Linderman, James Zou, and Christopher Re. 2025. Cost-efficient collaboration between on-device and cloud language models. In *ICLR 2025 Workshop on Foundation Models in the Wild*.

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

OpenAI. 2024. OpenAI o1 System Card. arXiv:2412.16720. ArXiv preprint.

Prasoon Patidar, Alex Crown, Kevin Hsieh, Yifei Xu, Tusher Chakraborty, Ranveer Chandra, and Yuvraj Agarwal. 2025. Orchestration for domain-specific edge-cloud language models. *arXiv preprint arXiv:2507.09003*.

Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Tamer Başar, and Ravishankar K Iyer. 2024. Power-aware deep learning model serving with {μ-Serve}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 75–93.

Margarita Reyes Sierra and Carlos A. Coello Coello. 2005. Improving pso-based multi-objective optimization using crowding, mutation and ϵ-dominance. In *Evolutionary Multi-Criterion Optimization (EMO 2005)*, volume 3410 of *Lecture Notes in Computer Science*, pages 505–519. Springer.

Wei Song, Zhenya Huang, Cheng Cheng, Weibo Gao, Bihan Xu, GuanHao Zhao, Fei Wang, and Runze Wu. 2025. IRT-router: Effective and interpretable multi-LLM routing via item response theory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15629–15644, Vienna, Austria. Association for Computational Linguistics.

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

Dimitris Stripelis, Zhaozhuo Xu, Zijian Hu, Alay Dilipbhai Shah, Han Jin, Yuhang Yao, Jipeng Zhang, Tong Zhang, Salman Avestimehr, and Chaoyang He. 2024. Tensoropera router: A multi-model router for efficient llm inference. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 452–462.

Tao Sun, Jian Xu, Yuanpeng Li, Zhao Yan, Ge Zhang, Lintao Xie, Lu Geng, Zheng Wang, Yueyan Chen, Qin Lin, and 1 others. 2025. Bitsai-cr: Automated code review via llm in practice. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, pages 274–285.

Yang Sun, Liangqing Wu, Shuangyong Song, Xiaoguang Yu, Xiaodong He, and Guohong Fu. 2022. Tracking satisfaction states for customer satisfaction prediction in e-commerce service chatbots. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 616–625.

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

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

Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, and 1 others. 2025b. Burstgpt: A real-world workload dataset to optimize llm serving systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 5831–5841.

Jiajun Xu, Zhiyuan Li, Wei Chen, Qun Wang, Xin Gao, Qi Cai, and Ziyuan Ling. 2024. On-device language models: A comprehensive review. *arXiv preprint arXiv:2409.00088*.

Shangzi Xue, Zhenya Huang, Jiayu Liu, Xin Lin, Yuting Ning, Binbin Jin, Xin Li, and Qi Liu. 2024. Decompose, analyze and rethink: Solving intricate problems with human-like reasoning cycle. *Advances in Neural Information Processing Systems*, 37:357–385.

Shibo Yu, Mohammad Goudarzi, and Adel Nadjaran Toosi. 2025. Efficient routing of inference requests across llm instances in cloud-edge computing. *arXiv preprint arXiv:2507.15553*.

Zesen Zhao, Shuowei Jin, and Z Morley Mao. 2024. Eagle: Efficient training-free router for multi-llm inference. *arXiv preprint arXiv:2409.15518*.

Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. 2023. Response length perception and sequence scheduling: An llm-empowered llm inference pipeline. *Advances in Neural Information Processing Systems*, 36:65517–65530.

A Experimental details

A.1 Candidate LLMs and their Pricing

We summarize the key characteristics of our candidate LLMs in Table 3, including model scale (parameter count), release date, model type, and token-based input/output pricing. To facilitate comparison and analysis, we categorize models into three tiers by size: Cost-Efficient LLMs ($<$ 7B) for ultra-low-cost, high-throughput settings; Capable Small LLMs (7B–20B) offering a more balanced trade-off between capability and cost; and Large LLMs ($\geq$ 20B) providing stronger reasoning and performance but typically at higher expense. This tiering helps routing/selection quickly align model choices with application-specific quality–efficiency requirements.

A.2 Dataset Details

Table 4 shows the details of datasets.

A.3 Prompt for metric collection

We use the following prompts to collect per-query results, including latency, correctness, and cost. As shown in Figure 8, we design three task-specific variants for code, math, and general multiple-choice questions. When crafting these prompts, we aim to elicit only the necessary outputs from the LLM and adopt a relatively structured format, which facilitates reliable parsing and automated extraction of the target fields.

B Predictor implementation details.

B.1 Feature Construction

For each query, we extract a hybrid feature vector that combines semantic, categorical, and scalar signals. Specifically, we use a Sentence-Transformer encoder all-MiniLM-L6-v2 to obtain a 384-dimensional query embedding. For categorical attributes (e.g., dataset kind and query difficulty), we adopt one-hot encodings. For scalar features (e.g., model parameter scale and input query length), we apply standard normalization. Finally, we concatenate the embedding, one-hot categorical features, and normalized scalars into a single tabular feature vector.

Table 3: Candidate LLMs and their relevant features.

Tier	Model	Params (B)	Release Date	Type	Input ($/1M)	Output ($/1M)
Large LLM ($\geq$20B)						
Large LLM	deepSeek-R1-671B	671.0	2025/01/20	reasoning	0.56	2.19
Large LLM	gemma3:27b	27.0	2025/03/12	general	0.04	0.15
Capable Small LLM (7B–20B)						
Capable Small LLM	llama3:8b	8.0	2024/04/18	general	0.03	0.06
Capable Small LLM	mathstral:7b	7.0	2024/07/16	math	0.15	0.15
Capable Small LLM	phi4:latest	14.0	2024/12/12	reasoning	0.06	0.14
Capable Small LLM	qwen2.5:7b-instruct	7.6	2024/11/22	general	0.04	0.10
Cost-Efficient LLM ($<$7B)						
Cost-Efficient LLM	deepseek-r1:1.5b	1.78	2025/01/20	reasoning	0.01	0.01
Cost-Efficient LLM	llama3.2:3b	3.0	2024/09/25	general	0.02	0.02
Cost-Efficient LLM	phi4-mini:latest	3.8	2025/03/11	reasoning	0.01	0.01
Cost-Efficient LLM	qwen2.5:3b-instruct	3.09	2024/11/22	general	0.02	0.02

Table 4: Datasets used in our evaluation. We report task type, evaluation metric, and the sizes of train/val/test splits for in-distribution and out-of-distribution settings.

In-distribution

Dataset	Task type	Metric	Train	Val	Test
ARC-Easy	Reasoning	accuracy	4157	519	521
GSM8K	Math	accuracy	7033	879	880
MBPP	Code	pass@1	771	96	97
MMLU	Multitask	accuracy	12686	1585	1587
MuSR	Multi-step Reasoning	accuracy	604	75	77

Out-of-distribution

Dataset	Task type	Metric	Train	Val	Test
CommonsenseQA	Reasoning	choice accuracy	–	–	2000
MATH	Math	pass@1	–	–	2000

Cold-start on unseen datasets. For out-of-distribution datasets, some features like query difficulty and dataset kind cannot be directly accessed. Here we predict dataset and q_difficult using KNN classifiers trained on seen data, and then feed the predicted labels into the one-hot pipeline. For dataset identification, KNN achieves an accuracy of 0.9689 with $n\_neighbors = 30$, cosine distance, and distance-weighted voting. For difficulty prediction, KNN yields Macro-F1$= 0.6849$ using $n\_neighbors = 25$ (also cosine distance with distance-weighted voting), indicating there is still room for improvement in difficulty modeling.

B.2 LLM Output Length Predictor

We model the output length as a regression problem. Given the constructed tabular feature vector, we adopt an XGBoost regressor due to its strong performance on mixed-feature tabular data. We use a moderately deep boosted tree ensemble with a large number of estimators and conservative learning rate to capture non-linear feature interactions while avoiding underfitting: max_depth=8, learning_rate=0.05, subsample=0.8, and colsample_bytree=0.8. We train with squared error objective and RMSE as the evaluation metric.

Comparison with a BERT baseline. To justify the design choice, we compare XGBoost against a BERT-based baseline that follows a standard setup: we encode the query using a BERT encoder and attach an MLP head to regress output length. Table 6 reports that XGBoost consistently achieves lower error and higher R^2. We attribute the gain to the fact that the tabular feature explicitly incorporates informative non-text signals, and allows XGBoost to exploit feature interactions that are highly predictive of output length.

Prompt

Respond in TWO parts:
1) On the FIRST line, print ONLY the final choice marker in the form ###<LETTER> (e.g., ###A). Do NOT add anything else on this line.
2) Starting from the SECOND line, provide your full reasoning/analysis step by step.

Prompt

You are a Python coding assistant.
Write Python code that solves the problem below
Output ONLY valid Python code (no markdown fences, no extra text)
The problem is as follows:
.....

Prompt

You are a competition mathematics assistant.
Solve the following problem step by step.
At the very end, on a separate line, output ONLY the final answer in the form ###ANSWER.
The problem is as follows:
.....

Figure 8: Prompts used for result extraction across task types: general multiple-choice, code, and math.

B.3 Relationship Between Output Length and End-to-end Latency

To reveal the quantitative relationship between output length and end-to-end latency, we estimate (k_m, b_m) using ordinary least squares on the training split for each model and report the goodness-of-fit on held-out data in Table 5. Across a wide range of models, the linear fit achieves strong R^2 (often > 0.8), suggesting output length is a dominant driver of latency variability.

Length-based latency estimation. Using the learned length–latency mapping, we further obtain an overall $R^2 = 0.5791$ when predicting end-to-end latency from (predicted) output length. Notably, this value is close to the R^2 of our output-length predictor (Table 6), which implies that improving length prediction accuracy can directly translate into more accurate latency estimation, consistent with our discussion in the limitations section.

Table 5: Per-model linear fit for latency–output length: $\text{Latency}_m \approx 10^{-2} \cdot k_m \cdot \text{OutputTokens} + b_m$.

Model	$R^2 \uparrow$	k (s/token)	b (s)
deepseekR1:1.5b	0.97	0.6	0.259
phi4-mini:latest	0.95	0.9	0.114
gemma3:27b	0.92	7.6	-0.539
qwen2.5:7b	0.89	1.2	0.093
llama3.2:3b	0.87	0.8	0.051
qwen2.5:3b	0.83	0.9	0.07
llama3:8b	0.82	1.6	0.197
mathstral:7b	0.82	1.2	-0.040
phi4:latest	0.82	2.1	-0.114
deepSeekR1:671b	0.71	8.3	-0.556

Table 6: Output-length prediction performance (lower MAE/MSE is better; higher R^2 is better).

Method	MAE ↓	MSE ↓	$R^2 \uparrow$
XGBoost	101.2	60193	0.6122
BERT + MLP	107.5	67530	0.5649

B.4 LLM Accuracy Predictor

We formulate accuracy prediction as a binary classification task and adopt an XGBoost classifier. We use a deep boosted tree ensemble with a conservative learning rate to capture non-linear feature interactions, with max_depth as 8 and learning_rate as 0.05; we also apply row/column subsampling to improve generalization.

Comparison with alternative predictors. We compare XGBoost with (1) a BERT-based classifier using the final hidden layer representation followed by an MLP head, and (2) a traditional KNN-based predictor used in prior routing pipelines. Table 7 shows that XGBoost provides the best overall discrimination, achieving the highest AUC and F1.

C Full Cold-start Results.

Due to space constraints, we report four representative models with different parameter scales, including deepseek-r1:1.5b, llama3.2:3b, llama3:8b, and gemma3:27b. Figure 9 summarizes their cold-start behavior in terms of accuracy (F1) and latency prediction error (MAE) as the profiling steps increase.

Overall, most models exhibit a clear stabilization trend around step $= 8$ (i.e., $\sim$200 profiling requests), after which both F1 and latency MAE change only marginally. At convergence, the cold-

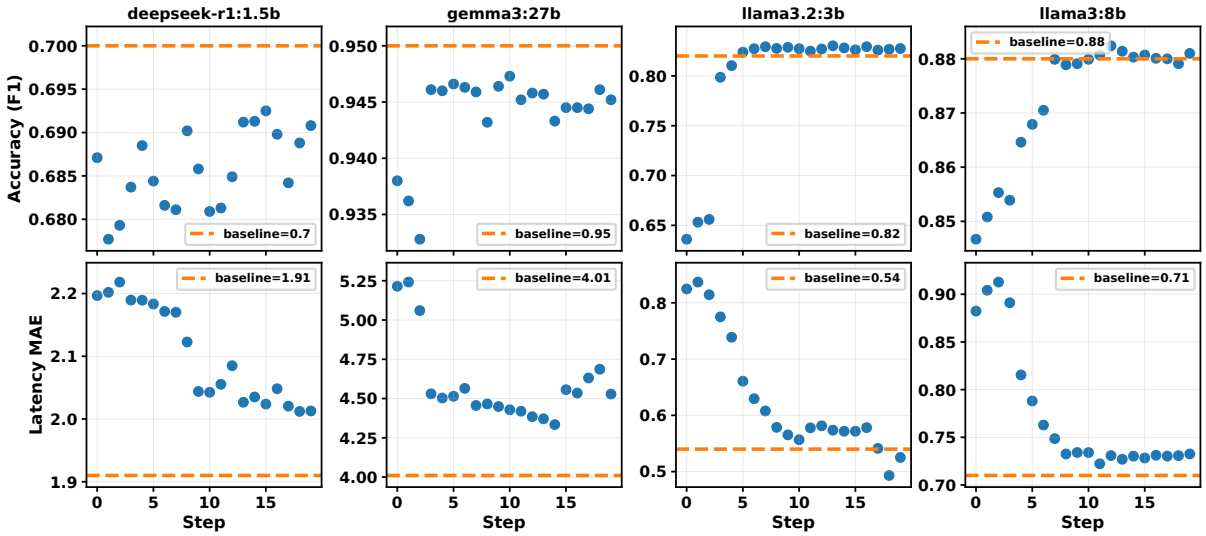

Figure 9: Full cold-start results on four representative models with different parameter scales. For each model, we report the accuracy predictor F1 and latency prediction MAE as profiling progresses. The x-axis shows the cold-start step. Horizontal dashed lines indicate the corresponding in-system reference of each model.

Table 7: Accuracy prediction performance (higher is better).

Method	ACC ↑	F1 ↑	AUC ↑
XGBoost (tabular)	0.8609	0.8856	0.9351
BERT + MLP	0.8511	0.8356	0.9254
KNN	0.6113	0.5627	0.7037

start performance is close to the corresponding in-system reference for the same model, with the gap typically within 10%. For cases where the curves do not show an obvious monotonic convergence (e.g., the F1 of deepseek-r1:1.5b), the discrepancy to the in-system reference is already small (below 4%) even at early steps. In addition, for models that start with a relatively large initial gap (e.g., llama3.2:3b), the cold-start procedure gradually closes the gap and reaches a level comparable to the in-system performance.

These results suggest that our cold-start strategy can effectively adapt to unseen LLMs with a small amount of profiling, and scales well across models of different sizes.

D Qualitative case study of routing decisions

To provide a more interpretable view of FLARE's query-level behavior, we present a qualitative case study with three representative queries that span different difficulty levels. Following the cross-LLM correctness statistics used in the main text, we treat larger query difficulty values as indicating that fewer candidate LLMs can solve the query correctly. For each example, we compare the Oracle choice, the LLM selected by FLARE-Balance, and a representative Cand-Random assignment.

Table 8 shows three typical patterns. First, for difficult reasoning-heavy queries, FLARE-Balance tends to route requests to the strongest LLM when weaker candidates are unlikely to satisfy the accuracy requirement. For example, gsm8k-97 has a difficulty score of 0.9 and requires multi-step numerical reasoning. In this case, FLARE-Balance selects deepSeek-R1-671B, matching the Oracle decision, whereas Cand-Random assigns the query to llama3:8b, which is more efficient but insufficiently capable for this query.

Second, for medium-difficulty knowledge questions, FLARE-Balance often selects a capable small LLM that preserves accuracy while avoiding unnecessary resource usage. For ACTAAP_2009_5_15 (difficulty 0.7), FLARE-Balance selects qwen2.5:7b, which also matches the Oracle decision. In contrast, Cand-Random chooses gemma3:27b, incurring higher latency and cost without a corresponding accuracy benefit.

Third, for easy interpretation-style questions, FLARE-Balance may select a smaller LLM than the Oracle choice while still maintaining the intended efficiency objective. For Mercury_402144 (difficulty 0.2), the Oracle selects llama3.2:3b, while FLARE-Balance chooses qwen2.5:3b. Although the selected LLM does not exactly match

Table 8: Representative query-level routing cases comparing Oracle, FLARE-Balance, and Cand-Random. Larger difficulty values indicate harder queries.

ID	Query	Diff.	Oracle	FLARE-Bal.	Random
gsm8k_97	Nancy is shelving books from the cart. She shelved 12 history books, 8 romance books, and 4 poetry books from the top section of the cart. Half the books on the bottom section of the cart were mystery books, which she quickly put back into place. Then, she shelved the remaining books from the bottom of the cart, including 5 Western novels and 6 biographies. How many books did she have on the book cart when she started?	0.9	deepSeek -R1-671B	deepSeek -R1-671B	llama3:8b
ACTAAP _2009_5_15	Question: Which is the best description of the brightness of the Sun? A. It is less bright than planets. B. It is brighter than other stars. C. It is of average brightness compared to planets. D. It is of average brightness compared to other stars.	0.7	qwen2.5:7b	qwen2.5:7b	gemma3:27b
Mercury _402144	Question: Look at the equation. $2H_2 + O_2 \rightarrow 2H_2O$. What does the equation show? A. heat being added to oxygen B. helium mixing with oxygen C. heat combining with oxygen D. hydrogen and oxygen combining	0.2	llama3.2:3b	qwen2.5:3b	mathstral:7b

Table 9: Hyperparameter settings for the three FLARE routing strategies.

Strategy	α	β	ω	τ
FLARE-Efficient	0.55	0.45	–	0.50
FLARE-Balance	0.4	0.25	0.35	0.50
FLARE-Acc	0.0	0.0	1.0	–
Oracle	0.5	0.3	0.25	–

the Oracle assignment, the decision remains consistent with the low-difficulty nature of the query and preserves the desired latency-cost trade-off. By contrast, Cand-Random routes the same query to mathstral:7b, which increases resource usage without improving answer quality.

Overall, these examples illustrate that FLARE-Balance produces query-dependent routing decisions aligned with the intended accuracy-efficiency trade-off: it escalates to stronger LLM for hard queries, prefers capable small LLMs for medium-difficulty cases, and uses lightweight LLM for easy queries whenever possible.

E Parameter Configuration.

E.1 Hyperparameter Choice.

Table 9 shows the final parameter choice for three FLARE modes over all datasets. We perform a grid search for the routing hyperparameters. Specifically, we enumerate (α, β, ω) on a 0.05 grid over $[0, 1]$ (inclusive) subject to $\alpha + \beta + \omega = 1$, and search the accuracy threshold τ on the same 0.05 grid over $[0, 1]$ (inclusive).

E.2 Sensitivity Analysis.

We perform a one-at-a-time sensitivity analysis on the routing hyperparameters: the candidate threshold τ and the utility weights (α, β, ω). For each run, we route all queries and report the mean Accuracy, Cost, and Latency.

When scanning the threshold τ, we set $\omega = 0$ so that accuracy affects routing only through candidate filtering, and we keep $\beta = \alpha = 0.5$. Candidates are defined as models whose predicted correctness probability satisfies $\hat{p}_m(x) \geq \tau$, and we select the model with the minimum utility score among candidates.

When scanning β or α, we fix $\tau = 0.5$ and set $\omega = \frac{1}{3}$. We vary one weight at a time and allocate the remaining weight mass to the other efficiency term so that α+β+ω=1 holds. When scanning ω, we fix $\tau = 0$ and set $\alpha = \beta = \frac{1-\omega}{2}$ to keep the weights normalized.

Figure 10 reveals clear and interpretable trends under one-at-a-time perturbations, highlighting two distinct mechanisms: (1) the threshold τ reshapes the candidate set, potentially triggering fallback behavior when candidates become scarce; and (2) the utility weights (α, β, ω) continuously steer the router's preference among efficiency and quality.(3) Notably, varying α and β induces only minor

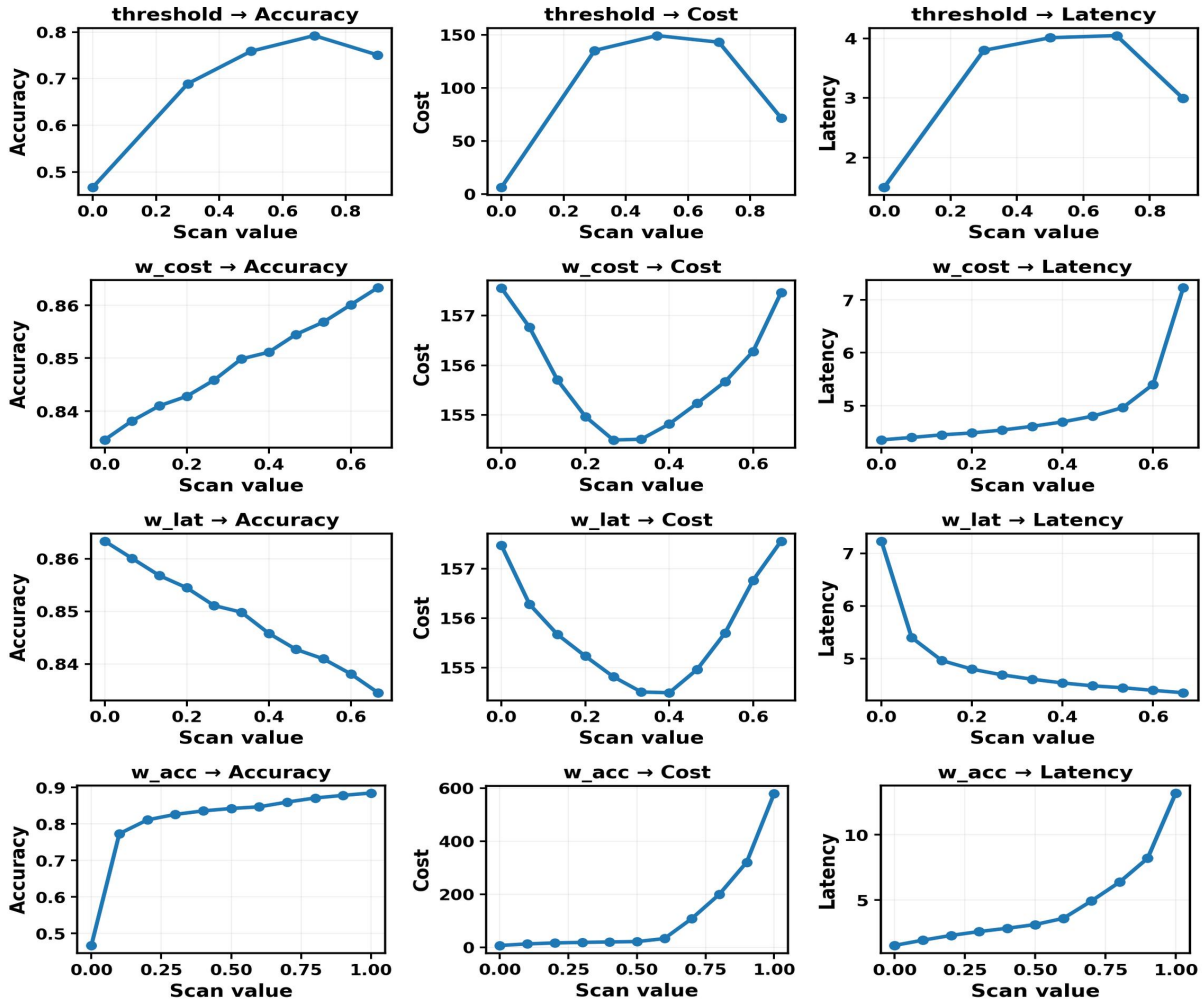

Figure 10: Sensitivity of routing performance to the candidate threshold τ and utility weights. Each row varies one parameter (τ, α,β, or ω), while each column reports the resulting mean Accuracy, Cost, and Latency. The x-axis shows the scanned value of the corresponding parameter, with all other settings held fixed.

changes (within $\pm 5\%$) in their corresponding efficiency metrics, suggesting that the router is robust in identifying near-optimal cost/latency options, whereas τ and ω cause substantially larger fluctuations in accuracy, indicating that the framework offers stronger and more direct control over the accuracy requirement and its trade-off with efficiency.

Threshold τ. As τ increases from low to moderate values, the mean Accuracy rises sharply, indicating that the probability-based filtering effectively removes low-confidence candidates and biases routing toward higher-quality models. In the same regime, both mean Cost and mean Latency increase, suggesting that the remaining feasible candidates tend to be more expensive and/or slower. When τ becomes high, however, Cost and Latency drop noticeably while Accuracy slightly decreases. This reversal is consistent with candidate scarcity: an overly strict threshold leaves fewer (or no) feasible candidates for many queries, increasing the frequency of falling back to the minimum-score choice over the full model set, which improves efficiency but sacrifices quality. Overall, τ exhibits a characteristic "sweet spot": too small yields insufficient quality control, while too large over-prunes candidates and weakens quality gains.

Varying α. Increasing α yields a strong, monotonic reduction in mean Latency with diminishing returns (rapid gains at small-to-moderate weights, followed by a flatter tail). In contrast, mean Accuracy decreases approximately monotonically, reflecting the expected trade-off when prioritizing faster models. Mean Cost again follows a U-shaped pattern: it is elevated when latency is underweighted (more expensive models may be chosen for quality or speed), and it rises again at high α

Table 10: Sensitivity of FLARE-Efficient and FLARE-Balance to controlled output-length prediction noise. We emulate different predictor qualities by injecting zero-mean noise into ground-truth lengths and report the resulting latency-cost-IGD trade-offs under fixed routing modes.

R^2	**FLARE-Efficient**			**FLARE-Balance**		
	Latency ↓	Cost ↓	IGD ↓	Latency ↓	Cost ↓	IGD ↓
1.0	3.67	148.98	0.06	4.57	154.89	0.08
0.9	5.00	155.33	0.11	5.74	159.55	0.13
0.8	5.66	158.18	0.14	6.28	161.75	0.16
0.7	6.09	160.93	0.16	6.63	164.04	0.17
0.6	6.45	165.04	0.17	6.94	168.00	0.19
0.5	6.69	168.22	0.18	7.17	171.07	0.20
0.4	6.92	171.51	0.19	7.36	174.89	0.21
0.3	7.10	175.26	0.20	7.53	178.12	0.22
0.2	7.25	179.70	0.21	7.67	182.79	0.22
0.1	7.39	184.38	0.22	7.79	187.03	0.23
0.0	7.51	187.51	0.22	7.90	190.13	0.24
-0.1	7.61	190.55	0.23	8.00	193.66	0.24
-0.2	7.72	194.83	0.23	8.11	197.97	0.25
-0.3	7.80	198.38	0.24	8.20	201.83	0.25
-0.4	7.90	202.57	0.25	8.29	206.55	0.26
-0.5	8.01	206.30	0.25	8.36	209.59	0.26
-0.6	8.09	209.42	0.26	8.44	214.24	0.27
-0.7	8.18	214.12	0.26	8.53	217.98	0.28
-0.8	8.26	218.17	0.27	8.61	222.96	0.28
-0.9	8.33	221.78	0.27	8.69	227.66	0.29
-1.0	10.47	464.80	0.31	10.47	464.80	0.31

where aggressively optimizing latency can push routing toward pricier fast models.

Varying β. With τ fixed, increasing β leads to a mild but consistent improvement in mean Accuracy, while mean Latency increases and becomes steep at the high-β end. This behavior aligns with the router increasingly favoring cheaper models that may be slower. Meanwhile, mean Cost shows a U-shaped curve: it is high when cost is underweighted (the router may select fast but expensive models), decreases around intermediate weights, and rises again when β is too large. The latter increase suggests that extreme cost emphasis interacts with candidate constraints and per-query normalization, yielding less stable choices that are not globally cost-optimal on average.

Varying ω. Increasing ω produces a rapid initial jump in mean Accuracy followed by saturation, indicating that most quality gains are realized once accuracy begins to influence ranking. In contrast, both mean Cost and mean Latency grow convexly and become steep at large ω, implying that aggressively prioritizing predicted correctness drives the router toward substantially more expensive and slower models. This highlights a practical regime in which modest ω achieves most of the attainable quality benefit, whereas very large ω incurs disproportionate efficiency penalties for limited additional accuracy.

F Robustness to Output-Length Prediction Error.

To assess how sensitive FLARE is to output-length prediction quality, we perform a controlled perturbation study in which zero-mean noise is injected into the ground-truth output lengths to emulate predictors with different effective R^2 values. We then

re-run FLARE-Efficient and FLARE-Balance under the same routing modes and report the resulting latency, cost, and inverted generational distance (IGD).

The results in Table 10 show a stable degradation trend rather than a cliff-like failure mode. Across the full R^2 sweep, latency, cost, and IGD change in a largely monotonic manner for both routing modes, indicating that moderate length-prediction errors do not abruptly destabilize routing behavior. Around our pratical operating point ($R^2 \approx 0.6$), improving the predictor to $R^2 = 0.9$ yields moderate gains for FLARE-Balance, with latency, cost, and IGD improving by 17.3%, 5.0%, and 31.6%, respectively. Conversely, degrading the predictor to a coarse mean-length proxy ($R^2 = 0$) worsens latency, cost, and IGD by 13.8%, 13.2%, and 26.3%, respectively.

Even under extreme corruption, FLARE remains competitive. When the length signal becomes purely noisy ($R^2 = -1$), FLARE-Efficient still reduces latency, cost, and IGD by 68.9%, 78.6%, and 76.8%, respectively, relative to always using `deepSeek-R1-671B`. This behavior is consistent with the design of FLARE: the accuracy-threshold filtering mechanism is independent of length prediction and continues to eliminate weak candidates when they are unlikely to satisfy the accuracy constraint.

The results also suggest that pushing output-length prediction toward oracle accuracy may have limited practical benefit. Although $R^2 = 1$ provides the best trade-off, even a strong predictor with $R^2 = 0.9$ still exhibits non-trivial degradation relative to oracle length. This observation is also consistent with prior serving literature, where output-length prediction typically achieves only moderate accuracy in realistic settings due to prompt-dependent uncertainty, multiple valid reasoning paths, decoding configurations, and termination variability. Overall, these results indicate that FLARE is robust to moderate length-prediction noise, and that the current predictor quality already provides a practical trade-off for resource-aware routing.