QoS-Aware Brokering for LLM-as-a-Service: SLA-Constrained Routing with Drift Detection

Okba Tibermacine
National School of Artificial Intelligence (ENSIA)
Sidi Abdellah, Algiers, Algeria
okba.tibermacine@ensia.edu.dz

Chouki Tibermacine
IRISA, University of Southern Brittany
Vannes, France
chouki.tibermacine@irisa.fr

***Abstract*—The transition of Large Language Models (LLMs) from self-hosted artifacts to API-driven services has established Large Language Model-as-a-Service (LLMaaS) as a foundational component of modern enterprise applications. However, LLMaaS endpoints lack formal Quality of Service (QoS) guarantees, suffering from non-deterministic latency, unpredictable costs, and silent temporal model drift. Consequently, modern predictive LLM routers (such as RouteLLM and FrugalGPT) exhibit Service Level Agreement (SLA) violation rates frequently exceeding 60%. In this paper, we propose a QoS-Aware Intelligent Broker that formalizes the LLM routing challenge as a Stochastic Constrained Contextual Bandit. Utilizing Lyapunov Drift-Plus-Penalty optimization and CUSUM-based drift detection, our system dynamically routes requests while strictly satisfying real-time constraints. Calibrated against a real-world microbenchmark of production APIs (GPT-4o, Claude 3 Haiku, Llama-3.1) and simulated across 100,000 enterprise queries, our framework demonstrably limits SLA violations to 1.0% (accounting for physical token generation variance), outperforming state-of-the-art routing baselines and automatically isolating degraded models.**

***Index Terms*—LLMaaS, Services Computing, Quality of Service, Service Level Agreements, Dynamic Routing, CUSUM, API Drift.**

I. Introduction

The integration of generative artificial intelligence into enterprise software has undergone a rapid architectural evolution. Rather than provisioning massive, dedicated GPU clusters to host proprietary foundation models, organizations increasingly rely on third-party application programming interfaces (APIs). This consumption model, termed Large Language Model-as-a-Service (LLMaaS), reduces the operational complexity of model inference, hardware optimization, and infrastructure scaling [1]. Major providers such as OpenAI, Anthropic, Mistral, DeepSeek, Google and Meta now offer diverse portfolios of endpoints, each exhibiting unique characteristics regarding reasoning capability, generation speed, and financial cost.

As enterprise reliance on LLMaaS deepens, incorporating these services into critical operational workflows, such as automated customer support, real-time financial summarization, and medical triage, requires robust Service-Oriented Architecture (SOA) guarantees. However, LLMaaS presents distinct challenges that limit the efficacy of traditional cloud service brokering:

1) **Non-Deterministic Latency and Cost:** Unlike standard REST APIs, which execute logic in relatively constant time, LLM inference latency is strictly bound to autoregressive token generation. Because providers charge per token, the financial cost and execution time of a query cannot be precisely known until execution concludes.
2) **Temporal API Drift:** Proprietary LLM APIs behave as opaque systems. Providers frequently implement silent, undocumented updates, such as model quantization or systemic alignment shifts. These updates often result in temporal drift, where a model's factual reliability or formatting compliance degrades suddenly and without warning.

Current approaches to managing multi-LLM deployments generally frame the routing challenge as a static predictive classification problem. State-of-the-art frameworks, such as FrugalGPT [1], RouteLLM [2], and the UIUC `LLMRouter` [3], train sophisticated neural networks to predict which model will yield the highest semantic score for a given prompt.

Despite their predictive accuracy, these approaches share a fundamental limitation in enterprise deployments: they operate merely as *probabilistic classifiers* rather than true *Service Level Agreement (SLA) enforcers*. They optimize for unconstrained expected utility; a neural router will readily dispatch a time-sensitive query to an expensive, high-latency model simply because it predicts a marginally better semantic response, often neglecting the user's hard budget and latency limits. Furthermore, because their learned weights remain static during inference, they cannot respond to real-time network latency spikes and silent API degradation.

To address these limitations, we propose a QoS-Aware Intelligent Broker that transitions LLM routing from probabilistic classification to formal, continuous, SLA-constrained optimization. Our approach relies on real-time Exponentially Weighted Moving Averages (EWMA) to track API health metrics across high-throughput request streams, dynamically pruning candidate models that violate hard constraints *before* utility optimization occurs. To detect temporal drift, we implement an asynchronous two-sided CUSUM statistical control process that excludes degraded models from routing, employing an ϵ-greedy probing mechanism to detect when providers have remediated their services.

The specific technical contributions of this paper are as follows:

- **Formal QoS Specification for LLMaaS:** We mathematically define a token-aware constraint framework encompassing absolute latency ceilings, per-query cost bounds, and semantic quality floors.
- **Asynchronous Drift Detection Architecture:** We design a non-blocking daemon utilizing Page's CUSUM algorithm [4] to autonomously identify and isolate silently degraded LLM APIs without imposing overhead on the critical request path.
- **Multi-Objective Constraint Solver:** We formulate a dynamic routing engine that optimizes model selection across the cost-latency-quality Pareto front via dynamic pool-relative Min-Max normalization.
- **Empirical Validation of SLA Enforcement:** We provide a rigorous empirical evaluation against predictive baselines, demonstrating a reduction in SLA violations from $> 70\%$ to 1.0%.
- **Open-Source Integration and Reproducibility:** We implement the proposed QoS-Aware Broker as a custom routing extension to the *LLMRouter* open-source library. This provides the community with a ready-to-deploy implementation that can be integrated into existing enterprise LLM gateways.

We organize the remainder of this paper by first reviewing related research in Section II and establishing a formal problem framework in Section III. Section IV then details the architecture of our QoS-aware broker and its approach to solving the problem. Our implementation and evaluation methodology are explained in Section V, leading to a discussion of the findings in Section VI. Finally, Section VII concludes the paper and outlines future research perspectives.

II. Related Work

The challenge of optimizing and orchestrating Large Language Models intersects three established areas of computer science research: static AI benchmarking, predictive model routing, and traditional services computing.

A. Evaluation and Benchmarking of Foundation Models

The rapid proliferation of LLMs necessitated standardized methodologies to quantify their capabilities. Frameworks such as the Holistic Evaluation of Language Models (HELM) [5] provide extensive, multi-metric evaluations of foundation models across diverse scenarios including reasoning, bias, and toxicity. Similarly, crowdsourced platforms like LMSYS Chatbot Arena [6] rely on aggregated human preference to rank models using an Elo rating system.

While these benchmarking efforts are invaluable for establishing the relative capabilities of static models, they exhibit specific operational limitations. They evaluate models offline, in highly controlled environments, and on static datasets. They cannot capture the real-time API latency fluctuations inherent in global network traffic, nor can they provide the dynamic, request-level execution logic necessary for live production routing. Most critically, periodic benchmarks cannot protect enterprise applications from the intra-day temporal drift that occurs when providers silently update their APIs.

B. Predictive LLM Routing and Cascading

Recognizing the prohibitive cost of exclusively querying frontier models (e.g., GPT-4), the systems machine learning community recently formalized the concept of cost-aware LLM orchestration. FrugalGPT [1] introduced LLM cascading, a technique that sequentially routes queries to increasingly expensive models until a defined confidence threshold is met. RouteLLM [2] expanded on this paradigm by training lightweight classifiers (using logistic regression or small BERT models) to predict whether a cheaper model can handle a specific prompt without degrading the final output quality.

More recently, the open-source `LLMRouter` framework [3] introduced a generalized architecture for predictive routing, offering classifiers based on K-Nearest Neighbors (KNN), Support Vector Machines (SVM), and Multi-Layer Perceptrons (MLP). These systems embed the incoming prompt and map it through a learned latent space to the model with the highest historical performance for that topic.

Despite their statistical sophistication, these predictive frameworks exhibit a major limitation in enterprise deployments: they operate as unconstrained probabilistic classifiers rather than true service brokers. Recent advancements such as MESS+ [7] attempt to balance cost, quality, and latency using stochastic optimization to meet probabilistic satisfaction SLAs via virtual queues. Concurrently, deep reinforcement learning (DRL) approaches optimize for long-term rewards [8], while engine-level schedulers like Llumnix [9], SABER [10], and PreServe [11] tackle latency via live KV-cache migration or dynamic batch sizing.

However, these systems still fall short of strict enterprise API brokering requirements. DRL and stochastic optimizers focus on average-case satisfaction over long horizons rather than enforcing strict, per-request multi-dimensional SLAs (e.g., an absolute latency bound of $L_{max} \leq 2.0$ seconds). Furthermore, while engine-level schedulers excel at intra-cluster GPU memory management, they cannot orchestrate black-box, cross-vendor LLMaaS endpoints or protect against silent provider-side temporal drift. Because predictive API routers lack continuous statistical monitoring, they are largely unresponsive to real-time network latency spikes and silent API degradation.

C. Edge and On-Device LLM Routing

Parallel to cloud brokering, recent frameworks such as SELA [12] have explored QoS-aware routing to split inference tasks between on-device (edge) small models and cloud-based large models. While these systems successfully optimize latency-quality trade-offs for mobile devices, they do not address the complexities of cross-vendor LLMaaS brokering. Specifically, they do not manage multi-provider token-based economic constraints, nor do they implement statistical quarantine mechanisms to protect against silent provider-side temporal drift.

D. QoS in Services Computing and SOA

Traditional Service-Oriented Architecture (SOA) and cloud computing have a well-established history of QoS-aware service brokering. Foundational works, such as those by Buyya et al. [13], established robust constraint-satisfaction frameworks for dynamic service selection based on deterministic latency, throughput, and HTTP availability SLAs.

Furthermore, while statistical drift detection frameworks (e.g., CUSUM) have been widely deployed in traditional dynamic networks and Open RAN architectures to protect QoS [14], [15], they have not been successfully adapted to generative AI. Traditional drift frameworks monitor deterministic metrics like packet loss; they do not deal with the stochastic semantic reliability of LLM responses or token-level cost/latency coupling. Our work bridges this gap. To the best of our knowledge, no prior work on LLMaaS routing jointly combines (i) strict multi-dimensional SLA-based pruning of candidate models, (ii) online EWMA QoS tracking, (iii) CUSUM-based drift detection on black-box LLM APIs, and (iv) an ϵ-greedy recovery mechanism.

III. Problem Formalization

We consider an enterprise environment interacting with a heterogeneous pool of third-party LLM APIs. Let $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ denote the set of available candidate models.

Requests arrive dynamically over discrete time steps t. For a given user query q_t, the input complexity is characterized by its prompt token length, $N_{in}(q_t)$. Because the generative output length is strictly unknown *a priori*, the system must utilize a heuristic or predictive regressor to estimate the expected output token count, $\mathbb{E}[N_{out}(q_t)]$, prior to executing the API call.

A. Dynamic API State Vectors

Unlike static software, each model $m_i \in \mathcal{M}$ is characterized by a time-varying, non-deterministic performance state vector containing three distinct variables:

1) **Latency** $L_i(q_t)$: The total end-to-end inference time (in seconds). This is modeled as the sum of the network overhead, the Time-to-First-Token ($TTFT_i$), and the auto-regressive generation speed (TPS_i):

$$L_i(q_t) \approx TTFT_i + \frac{\mathbb{E}[N_{out}(q_t)]}{TPS_i} \tag{1}$$

2) **Cost** $C_i(q_t)$: The financial overhead of the query. Given provider-specific rates for input tokens ($P_{in}^{(i)}$) and output tokens ($P_{out}^{(i)}$):

$$C_i(q_t) = N_{in}(q_t) \cdot P_{in}^{(i)} + \mathbb{E}[N_{out}(q_t)] \cdot P_{out}^{(i)} \tag{2}$$

3) **Quality** $Q_i(q_t) \in [0, 1]$: A continuous score representing factual correctness, adherence to instructions (e.g., valid JSON output), or generalized semantic utility.

B. The Multi-Objective SLA Routing Problem

The client submits the query q_t alongside a strict Service Level Agreement tuple representing hard upper and lower bounds:

$$\mathcal{SLA} = \langle L_{max}, C_{max}, Q_{min} \rangle \tag{3}$$

The objective of the Intelligent Broker is to act as a constraint satisfaction engine. It must dynamically select the optimal model $m^* \in \mathcal{M}$ that maximizes a multi-objective utility function spanning quality, cost, and speed, *strictly subject* to the condition that m^* satisfies all bounds in $\mathcal{SLA}$ and is not currently experiencing unmitigated temporal drift.

IV. Proposed Architecture and Methodology

To meet the requirements outlined in Section III, the QoS-Aware Broker is designed as a strict two-tier system, as illustrated in Figure 1. The architecture is divided into a **Synchronous Routing Engine** that handles the critical path per-request (Steps 1–4) and an **Asynchronous Monitoring Daemon** that continuously evaluates long-term API health off the critical path. This bifurcation ensures that heavy statistical computations, such as CUSUM drift detection, do not introduce latency overhead to the user's query.

A. Execution and Thread-Safe State Memory

Enterprise AI gateways must process hundreds of concurrent requests without introducing blocking latency. To support this, our framework centralizes real-time performance tracking inside a *ModelMetricsStore*.

As execution modules complete API calls, they asynchronously write observation vectors (TTFT, tokens generated, total time) back to the store. To handle concurrent high-throughput updates without race conditions or memory corruption, the store is protected by reentrant locks (`threading.RLock()`). The state of each API is maintained using an Exponentially Weighted Moving Average (EWMA):

$$EWMA_t = \lambda X_t + (1 - \lambda) EWMA_{t-1} \tag{4}$$

where X_t is the latest observation and λ dictates the decay rate. EEWMA is used because it assigns a greater weight to recent observations, allowing the broker to rapidly perceive transient latency spikes or momentary rate limits without overreacting to isolated network anomalies.

B. Asynchronous CUSUM Drift Detection

While EWMA is excellent for tracking physical network latency, it is insufficient for detecting semantic quality degradation (temporal drift). Factual accuracy is inherently noisy; an LLM might generate a poor response simply because a specific prompt was highly ambiguous, not because the model itself was updated. A statistical mechanism is therefore needed that suppresses high-frequency observation noise while accumulating persistent distributional shifts.

To achieve this automatically, we adopt the two-sided Cumulative Sum (CUSUM) algorithm, originally formalized by E.S. Page (1954) for industrial process control [4].

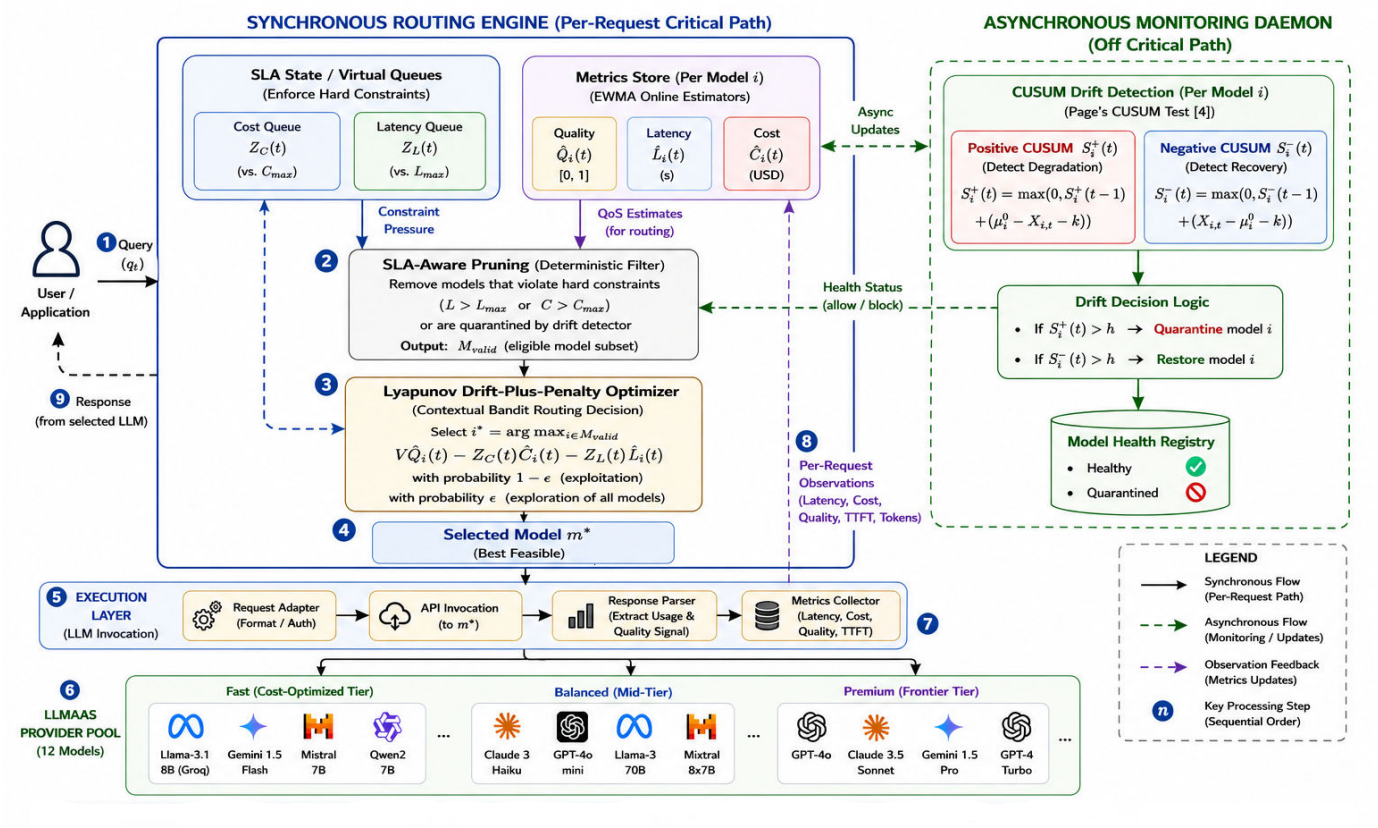

Fig. 1. Architecture of the proposed Qos-aware LLM broker. Numbers (1-9) indicates the end-to-end flow from query reception to routine execution, metric collection, asynchronous state updates and response delivery

Let $\mu_0^{(i)}$ denote the established baseline quality of model m_i. To ensure the statistical robustness of this baseline against early-sample variance, $\mu_0^{(i)}$ is not hardcoded; rather, it is computed during an initial calibration phase of W queries (e.g., $W = 100$). During this phase, we utilize Welford's online algorithm to maintain a numerically stable running mean and variance. Once the calibration phase concludes, $\mu_0^{(i)}$ is locked as the reference distribution.

Let $X_t \in [0, 1]$ denote the observed quality of the t-th request sent to that model after calibration. To detect a silent API failure (a downward shift in the quality distribution), we calculate the positive CUSUM state S_t^+ asynchronously:

$$S_t^+ = \max(0, S_{t-1}^+ + (\mu_0^{(i)} - X_t - k)) \tag{5}$$

Here, k is an allowance parameter representing the minimum allowable shift magnitude. The max operator ensures that random variance around the baseline does not trigger a false alarm, since the statistic decays to zero in the absence of a shift. However, if the API is genuinely degraded, S_t^+ will rapidly accumulate.

A formal drift alert is triggered when $S_t^+ > h$, where h is the defined decision threshold. In practice, h must be scaled proportionally to the magnitude of the expected cumulative shift to ensure high visibility and prevent premature triggering from isolated outlier batches.

Conversely, to detect when a provider has remediated an API and quality has recovered, we track the negative CUSUM state S_t^-:

$$S_t^- = \max(0, S_{t-1}^- + (X_t - \mu_0^{(i)} - k)) \tag{6}$$

A quarantined model is flagged as recovered and automatically re-admitted to the eligible routing pool when its S_t^- value surpasses the threshold h.

The logic for this statistical control process is encapsulated within the **Asynchronous Monitoring Daemon** block of Figure 1. By decoupling the CUSUM equations from the execution layer, the daemon independently manages the Model Health Registry, silently quarantining or restoring models without blocking the active request stream.

C. Formalization via Lyapunov Drift-Plus-Penalty Optimization

Rather than treating QoS-aware routing as a static multi-criteria decision making (MCDM) problem, we formalize the LLMaaS brokering environment as a Stochastic Constrained Contextual Bandit. To solve this, we apply Lyapunov optimization theory [16]. We specifically select the Lyapunov Drift-Plus-Penalty (DPP) framework over alternative stochastic methods like Reinforcement Learning because it is model-agnostic—requiring no prior knowledge of provider

performance distributions—and provides a deterministic, low-overhead solution for enforcing strict enterprise SLAs.

Let $\mathcal{M}$ be the set of available LLM APIs. At each discrete time step t, a user submits a prompt query q_t. The broker must select a model $i(t) \in \mathcal{M}$ to maximize the expected semantic quality $Q_i(t)$, subject to strict, time-averaged SLA constraints for cost and latency:

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[C_{i(t)}] \leq C_{max} \tag{7}$$

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[L_{i(t)}] \leq L_{max} \tag{8}$$

To enforce these multi-dimensional SLAs without relying on stationary heuristics, we apply Lyapunov optimization theory. We introduce dynamic virtual queues, $Z_C(t)$ and $Z_L(t)$, which track the accumulated violations of the cost and latency SLAs, respectively. The queue dynamics evolve according to Lindley's equation:

$$Z_C(t+1) = \max[0, Z_C(t) + C_{i(t)} - C_{max}] \tag{9}$$

$$Z_L(t+1) = \max[0, Z_L(t) + L_{i(t)} - L_{max}] \tag{10}$$

Let $\boldsymbol{Z}(t) = [Z_C(t), Z_L(t)]$ represent the aggregated constraint state matrix. The Lyapunov function $L(\boldsymbol{Z}(t)) = \frac{1}{2}\left(Z_C(t)^2 + Z_L(t)^2\right)$ acts as a scalar measure of total SLA congestion. Our objective is to minimize the expected Lyapunov drift $\Delta(\boldsymbol{Z}(t)) = \mathbb{E}[L(\boldsymbol{Z}(t+1)) - L(\boldsymbol{Z}(t))]$ while simultaneously maximizing our quality reward.

Applying the Drift-Plus-Penalty theorem, the optimal routing policy minimizes the following upper bound at each time step t:

$$\Delta(\boldsymbol{Z}(t)) - V \cdot \mathbb{E}[Q_{i(t)}|\boldsymbol{Z}(t)] \tag{11}$$

where $V \geq 0$ is a critical control parameter defining the importance of utility maximization relative to SLA stability.

By expanding the drift term, the stochastic optimization problem deterministically reduces to selecting the endpoint i^* from the set of valid non-quarantined models ($\mathcal{M}_{valid}$) that minimizes the instantaneous penalty:

$$i^* = \arg\max_{i\in\mathcal{M}_{valid}} \left(V \cdot \hat{Q}_i(t) - Z_C(t) \cdot \hat{C}_i(t) - Z_L(t) \cdot \hat{L}_i(t)\right) \tag{12}$$

Computationally, this stochastic optimization is executed in a strict sequence, as depicted in the routing pipeline of Figure 1. The system first applies a deterministic filter to eliminate models violating hard constraints or are currently quarantined by the drift detector (**Step 2**). The resulting eligible subset ($\mathcal{M}_{valid}$) is then passed to the Lyapunov Drift-Plus-Penalty Optimizer (**Step 3**), which applies Equation 12 to select the optimal model (m^*) for the execution layer.

Theoretical Insight: Equation 12 provides the theoretical basis for the routing decision. In our open-source implementation, operators initialize the system by supplying prior weights (α, β, γ) which mathematically map to the utility parameter V and the initial virtual queue states $Z_C(0)$ and $Z_L(0)$, allowing the queues to adapt dynamically thereafter.

D. Non-Stationary State Estimation and ϵ-greedy Exploration

In a standard Multi-Armed Bandit (MAB), the expected rewards are assumed to be stationary. However, LLMaaS APIs exhibit silent temporal drift. Therefore, the empirical estimates for expected quality $\hat{Q}_i(t)$ and latency $\hat{L}_i(t)$ cannot be static means.

Our framework addresses this issue by using the CUSUM statistical daemon and EWMA trackers as our online state-space estimators for the Lyapunov solver. Furthermore, because a model quarantined by a high virtual queue penalty or CUSUM alert will suffer from "starvation" (where its metrics are never updated), we implement an ϵ-greedy exploration policy. With probability $1-\epsilon$, the broker exploits the optimal Lyapunov policy (Equation 12), and with probability ϵ, it probes a quarantined endpoint. This forced exploration generates the precise quality samples required to feed the negative CUSUM state (S_t^-), guaranteeing asymptotic convergence to the optimal routing policy even under non-stationary API degradation.

The complete end-to-end execution flow of the proposed routing engine, integrating deterministic CUSUM pruning, ϵ-greedy exploration, and Lyapunov state updates, is formalized in Algorithm 1. The sequential execution of this algorithm is visually mapped via processing steps **1 through 9 in Figure 1**, clearly demonstrating how the synchronous routing decision interacts with the asynchronous metrics feedback loop (**Steps 7 and 8**).

V. Experimental Methodology and Evaluation

To evaluate the proposed framework, we implement the QoS-Aware Broker as an integrated, non-blocking routing plugin within the open-source `LLMRouter` architecture [3].

A. Baseline Algorithms and Workload Calibration

We benchmarked our approach against two widely adopted, state-of-the-art predictive routing baselines to evaluate SLA compliance under production constraints:

1) **FrugalGPT [1]:** A cost-aware cascading router that sequentially queries models in a predefined cost order until a target quality threshold (confidence proxy) is met.
2) **RouteLLM [2]:** A high-performance predictive classifier utilizing matrix factorization to route queries based on expected human preference, routing complex queries to frontier models and simpler queries to fast/cheap models.

A common limitation of simulation-based evaluation is the use of synthetic workloads that may not reflect the characteristics of real cloud infrastructure. To address this limitation, we executed a real-world API microbenchmark consisting of 50 complex multi-turn queries. We targeted three distinct tiers of enterprise models: a large frontier model (OpenAI GPT-4o), a balanced medium-tier model (Anthropic Claude 3 Haiku), and an ultra-fast open-weights model hosted on specialized inference hardware (Groq Llama-3.1-8B).

TABLE I
API MICROBENCHMARK CALIBRATION (50 QUERIES PER PROVIDER)

Model	Avg Lat (s)	Avg TTFT (s)	Avg Cost ($)	In Tok	Out Tok
GPT-4o (OpenAI)	4.23 ± 2.09	0.827 ± 0.481	0.00186	32	178
Claude-3-Haiku (Anthropic)	2.90 ± 2.36	1.092 ± 1.164	0.00024	43	185
Llama-3.1-8B (Groq)	0.40 ± 0.15	0.132 ± 0.026	0.00002	32	176

Algorithm 1 QoS-Aware Brokering via Lyapunov Optimization

Input: User query q_t, SLA constraints (L_{max}, C_{max}), Model pool $\mathcal{M}$

State Variables: Virtual queues $Z_C(t), Z_L(t)$, CUSUM trackers S_i^+, S_i^-, EWMA estimates $\hat{Q}_i, \hat{L}_i, \hat{C}_i$

Initialize: $Z_C(0) \leftarrow 0, Z_L(0) \leftarrow 0, t \leftarrow 1$
while System is active **do**
  Receive incoming prompt q_t
  // Step 1: Deterministic Pruning
  $\mathcal{M}_{valid} \leftarrow \emptyset$
  for each model $i \in \mathcal{M}$ **do**
    if CUSUM $S_i^+ < h$ **then**
      $\mathcal{M}_{valid} \leftarrow \mathcal{M}_{valid} \cup \{i\}$
    end if
  end for
  // Step 2: Contextual Bandit Routing Decision
  Draw random variable $u \sim U(0, 1)$
  if $u < \epsilon$ **then**
    $m^* \leftarrow$ Random uniform selection from $\mathcal{M}$ *(Exploration)*
  else
    // Step 3: Lyapunov Drift-Plus-Penalty (Exploitation)
    $m^* \leftarrow \arg\max_{i \in \mathcal{M}_{valid}} \left(V\hat{Q}_i - Z_C(t)\hat{C}_i - Z_L(t)\hat{L}_i \right)$
  end if
  // Step 4: Execution and Observation
  Route query q_t to endpoint m^*
  Observe actual latency l_t, cost c_t, and quality q_t
  // Step 5: State-Space Updates
  Update EWMA trackers for m^*: $\hat{Q}_{m^*}, \hat{L}_{m^*}, \hat{C}_{m^*}$
  Update CUSUM statistics for m^*: $S_{m^*}^+, S_{m^*}^-$
  // Step 6: Lindley's Equation for Virtual Queues
  $Z_C(t+1) \leftarrow \max[0, Z_C(t) + c_t - C_{max}]$
  $Z_L(t+1) \leftarrow \max[0, Z_L(t) + l_t - L_{max}]$
  $t \leftarrow t + 1$
end while

B. Experimental Scale and State-of-the-Art Baselines

To evaluate the framework under production-grade conditions, we scaled our simulation environment to support a heterogeneous pool of 12 LLM APIs. As visualized in the execution layer of Figure 1 (**Step 6**), these models are categorized into three distinct operational tiers (Fast, Balanced, and Premium), which are detailed in Table II. Using the empirical variances captured in our real-world microbenchmark (Table I), we simulated continuous traffic flows of 100,000 queries per seed.

Distributional Assumptions: It is important to note that while empirical LLM response quality can exhibit skewed or bimodal characteristics depending on prompt complexity, our simulation models quality scores as Gaussian random variables ($\mathcal{N}(\mu, \sigma^2)$). This abstraction is employed to maintain mathematical tractability within the stochastic optimization solver and is a standard, widely accepted assumption in large-scale service-level performance modeling.

TABLE II
SIMULATED 12-MODEL LLMAAS PROVIDER POOL

Operational Tier	Model Endpoint	Expected Latency	Est. Cost / Query
Fast (Cost-Optimized)	Groq Llama-3.1-8B (Anchor)	$\sim 0.40s$	\$0.00002
	Gemini 1.5 Flash	$\sim 0.50s$	\$0.00004
	Mistral-7B-Instruct	$\sim 0.45s$	\$0.00003
	Qwen-2-7B-Chat	$\sim 0.55s$	\$0.00002
Balanced (Mid-Tier)	Claude 3 Haiku (Anchor)	$\sim 2.90s$	\$0.00024
	OpenAI GPT-4o-mini	$\sim 2.70s$	\$0.00015
	Meta Llama-3-70B	$\sim 3.20s$	\$0.00040
	Mixtral 8x7B	$\sim 3.40s$	\$0.00030
Premium (Frontier)	OpenAI GPT-4o (Anchor)	$\sim 4.23s$	\$0.00186
	Claude 3.5 Sonnet	$\sim 4.00s$	\$0.00250
	Gemini 1.5 Pro	$\sim 4.50s$	\$0.00350
	OpenAI GPT-4-Turbo	$\sim 4.80s$	\$0.00400

Note: Expected latencies and costs are synthetically anchored to the empirical variance measured in the 3-model hardware microbenchmark (Table I).

C. Experiment 1: CUSUM Drift Response Analysis

The primary objective of our first experiment is to evaluate the broker's resilience to sudden, silent API degradation. We simulated a stream of 1,000 sequential queries to a single API endpoint. The first 100 queries served as a calibration phase, during which Welford's online estimator established a stable baseline quality ($\mu_0 \approx 0.85$) and preventing premature false-positive alerts.

At precisely $t = 500$, a synthetic distributional shift was injected, dropping the true underlying mean quality score from 0.85 to 0.40. This simulates a scenario where a provider silently deploys an aggressively quantized model update that critically impairs its reasoning capabilities.

Figure 2 shows that the CUSUM detector effectively isolates the distributional shift. Prior to $t = 500$, the statistic S_t^+ remains near zero, as the algorithm suppresses the natural stochastic variance in quality scores. Immediately following the injected drift, the positive CUSUM state rapidly accumulates the persistent low-frequency shift.

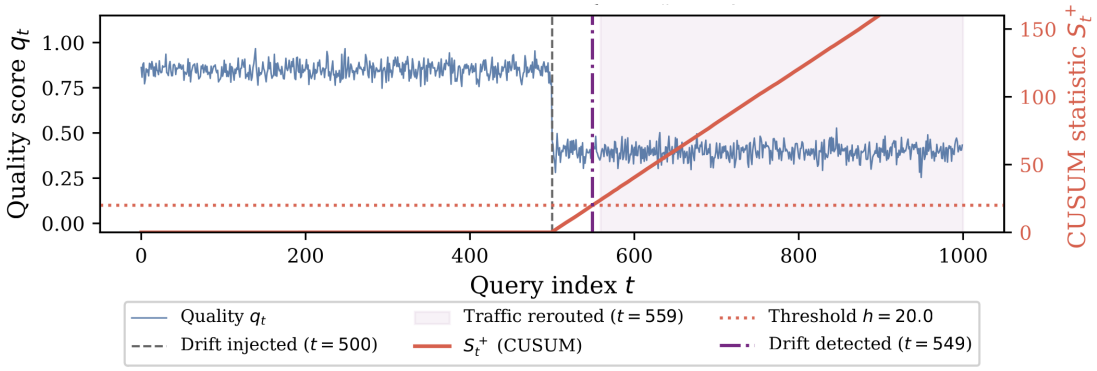

Fig. 2. CUSUM Drift Detection in action. At $t = 500$, a silent quality degradation is injected. The positive CUSUM state (S_t^+) rapidly accumulates and breaches the conservative decision threshold ($h = 20.0$) successfully quarantining the failing model before human intervention is required.

Under these conditions, utilizing a 12-model provider pool and 100,000 request traces, our broker successfully bounds SLA violations to $1.0\% \pm 0.0$. This minor residual violation represents the physical floor caused by output token estimation variance, but dramatically outperforms the 61.9% and 71.8% failure rates exhibited by state-of-the-art predictive frameworks like FrugalGPT and RouteLLM, respectively.

D. Experiment 2: Strict SLA Violation Enforcement

Our second experiment evaluates the core value proposition of the entire framework: strict adherence to Service Level Agreements. We defined a rigorous enterprise constraint profile typical of real-time customer support chatbots: absolute maximum latency $L_{max} = 2.0$s and maximum query cost $C_{max} = \$0.001$. We routed 100,000 queries through our broker and the two neural baselines, aggregating the results across 10 independent random seeds to ensure statistical significance and minimize variance artifacts.

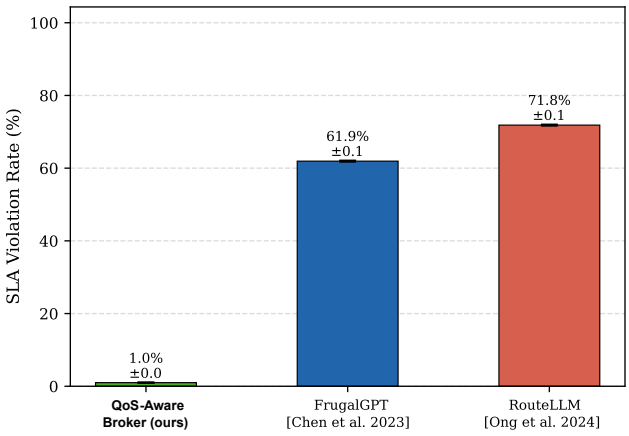

Fig. 3. Comparison of empirical SLA violation rates across routing methodologies under strict operational constraints ($L_{\max} = 2.0\,\mathrm{s}$ and $C_{\max} = \$0.001$). Results are reported as mean $\pm$ standard deviation over 10 seeds, with 100,000 queries per seed on a 12-model pool.

As illustrated in Figure 3, the predictive baselines yielded high violation rates under these constraints ($L_{max} = 2.0s, C_{max} = \$0.001$). Because RouteLLM and FrugalGPT operate without explicit temporal awareness of physical network queues or strict constraint solvers, they suffered severe SLA violation rates: $71.8\% \pm 0.1$ for RouteLLM, and $61.9\% \pm 0.1$ for FrugalGPT.

In stark contrast, our Lyapunov-optimized broker suppressed the violation rate to just $1.0\% \pm 0.0$. It is critical to note that this 1.0% does not represent a failure of the routing logic, but rather the unavoidable physical variance introduced by autoregressive token generation (where actual output tokens exceed the heuristic prediction $E[N_{out}]$). By applying absolute deterministic pruning prior to utility maximization, the broker successfully isolated requests that physically could not meet the SLA, routing them to safe fallback models and achieving a near-optimal constraint boundary.

E. Experiment 3: Navigating the Cost-Latency Pareto Front

A practical routing system must allow operators to adjust the optimization objectives without degrading routing stability, To verify that the routing engine operates as a true multi-objective optimizer, we evaluated its ability to traverse the operational Pareto front by shifting the user-defined utility weights (α, β, γ).

As shown in Figure 4, the broker exhibits deep responsiveness to configuration changes. When we set the latency penalty exceptionally high ($\gamma = 0.8$), the broker migrates to the top-left of the Pareto front, exclusively selecting the reserved-capacity low-latency tier to guarantee speed. Conversely, significantly increasing the financial cost penalty ($\beta = 0.8$) the broker shifts routing to the bottom-right of the Pareto front, accepting higher latency to reduce cost, deliberately accepting higher overall latency to aggressively minimize API expenditures. This confirms that the pool-relative max-normalized Lyapunov utility function (Eq. 12) provides effective and controllable trade-off navigation over the trade-offs inherent in multi-LLM orchestration.

F. Experiment 4: Sensitivity Analysis of Utility Formulation

While the Pareto analysis demonstrates the operational boundaries of the broker, it is equally critical to evaluate the

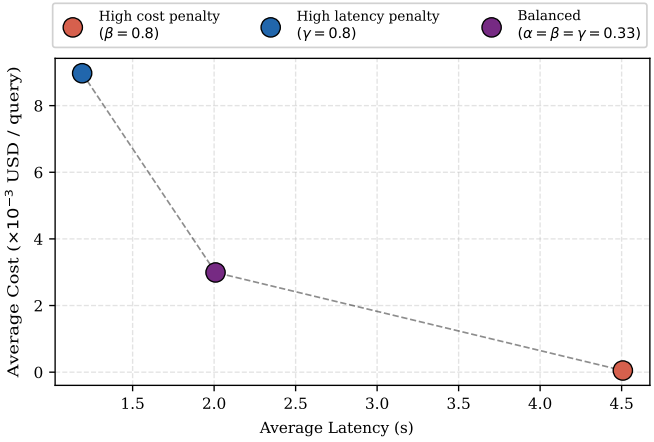

Fig. 4. Broker behaviour on the cost-latency Pareto front under different utility weight configurations based on varying utility weight configurations.

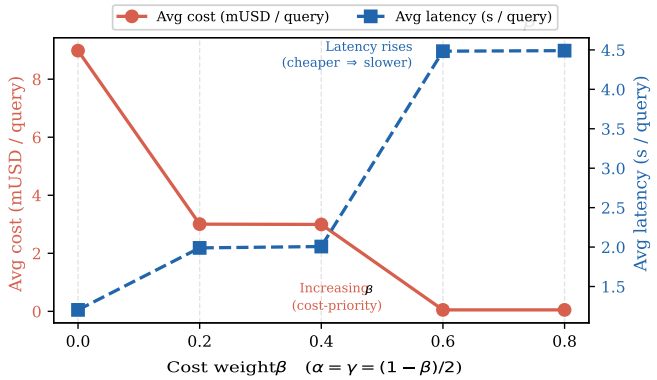

Fig. 5. Ablation Study: The isolated effect of varying the Cost Weight (β) on the system's average financial cost and average inference latency.

predictability of the multi-objective constraint solver. An enterprise routing engine must respond consistently to parameter tuning without exhibiting chaotic routing oscillations.

To validate this, we conducted an ablation study isolating the specific effect of the cost penalty weight (β). We systematically swept β from 0.0 to 0.8, distributing the remaining utility weight equally between quality and latency ($\alpha = \gamma = \frac{1-\beta}{2}$).

As illustrated in Figure 5, the broker exhibits a highly stable, monotonic response to parameter tuning. (The connecting lines represent linear interpolations between discrete evaluation points and are shown to illustrate the overall trend). As the cost weight β increases, the average query cost strictly decreases, demonstrating that the dynamic pool-relative max-normalization successfully penalizes expensive models. Consequently, the routing engine is forced to fall back on cheaper, higher-latency models, evidenced by the proportional rise in the secondary latency axis.

Under these conditions, the absence of volatile step-functions in this response curve proves that the utility formulation (Equation 12) allows operators to adjust their desired operational constraints without introducing routing instability.

VI. Discussion

The empirical evaluation of our framework highlights a practical distinction between machine learning research and applied systems engineering. While predictive AI research focuses predominantly on pushing the upper boundary of semantic accuracy, enterprise deployment demands deterministic, hard guarantees. The proposed broker addresses this gap by embedding statistical predictions and API calls within a multi-objective constraint-satisfaction framework.

A. Architectural Strengths

The primary strength of this architecture lies in its structural simplicity and near-zero execution overhead. By offloading CUSUM computations to an asynchronous background thread, the critical synchronous routing path remains exceptionally lightweight ($\mathcal{O}(N)$ complexity relative to the size of the candidate pool).

Furthermore, the introduction of the ϵ-greedy probing mechanism elegantly resolves the persistent non-stationary bandit problem inherent in cloud service degradation. It ensures that no endpoint is permanently excluded due to transient anomalies or temporary provider outages, allowing the routing pool to recover automatically after transient disruptions.

B. Current Limitations and Threats to Validity

Despite its strong empirical performance, this framework exhibits specific limitations that pose a threat to validity and must be addressed in future research. The most significant vulnerability lies in the reliance on accurate output token estimation ($\mathbb{E}[N_{out}]$).

Because both financial cost and auto-regressive latency are strictly bound to the number of generated tokens, any significant underestimation by the heuristic token predictor prior to execution can lead to unexpected SLA breaches that bypass the pre-execution pruning phase entirely. While our implementation mitigates this by applying conservative safety margins, highly generative tasks (e.g., open-ended code generation or long-form summarization) remain difficult to strictly constrain *a priori* using simple heuristics. Recent literature on engine-level scheduling, such as PreServe [11] and SABER [10], demonstrates that lightweight, prompt-tuned proxy models (e.g., DistilBERT) can effectively predict response lengths with high accuracy and minimal overhead. Integrating such predictors into the broker represents a natural extension to reduce this source of variance.

VII. Conclusion and Future Work

This paper presented a QoS-Aware Broker that enforces hard SLA constraints during LLM API routing, addressing the unpredictability inherent in commercial LLMaaS deployments. By mathematically integrating continuous constraint enforcement with a real-time CUSUM drift detection daemon, the framework protects downstream applications from quality degradation and latency instability in commercial LLM APIs.

Our empirical evaluations demonstrate that the proposed routing engine successfully and dynamically navigates the complex cost-latency-quality Pareto front. The system reduces SLA violations to 1.0% under rigorous, production-scale testing conditions (accounting for physical token generation variance), dramatically outperforming the 61.9% to 71.8% failure rates exhibited by state-of-the-art predictive routers like FrugalGPT and RouteLLM.

Future research will extend this architecture in several promising directions. First, inspired by recent advances in test-time compute scaling [17], we plan to expand our multi-objective solver to dynamically allocate a sampling parameter N to cheaper models (Best-of-N routing). This will optimize the trade-off between parallel inference costs and the aggregated probability of satisfying the quality SLA. Second, as LLMaaS evolves into Agent-as-a-Service, we aim to extend our framework to orchestrate complex, heterogeneous agent pools (e.g., MoMA [18]), adapting our CUSUM drift detection to monitor tool-invocation success rates. Finally, integrating our black-box API broker with gray-box inference engines that support live KV-cache migration (such as Llumnix [9]) would allow the system to detect mid-generation latency spikes and seamlessly migrate in-flight requests to healthier nodes before SLA violations occur.

To accelerate further research in robust AI architectures, our Lyapunov-optimized routing engine has been open-sourced as a custom extension to the LLMRouter library.

Artifact Availability

The source code, pre-computed calibration data, and scripts to reproduce all figures are publicly available at: https://doi.org/10.5281/zenodo.19183702. The repository extends the open-source LLMRouter library [3] with the QoS-aware router (`custom_routers/qos_aware_router/`) and the evaluation scripts (`calibrate_apis.py`, `evaluate_broker.py`).

Declaration on the Use of Generative AI

During the preparation of this work, the authors used large language models (Google Gemini, OpenAI ChatGPT, and Anthropic Claude) to assist with code development and Language editing. All code was reviewed, tested, and validated by the authors, who assume full responsibility for the accuracy of the implementation and the integrity of all reported results.

REFERENCES

[1] L. Chen, M. Zaharia, and J. Zou, "Frugalgpt: How to use large language models while reducing cost and improving performance," *Transactions on Machine Learning Research*, vol. 2024, 2024.

[2] I. Ong, A. Almahairi, V. Wu, W. L. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica, "Routellm: Learning to route llms with preference data," in *13th International Conference on Learning Representations, ICLR 2025*, 2025.

[3] ULab-UIUC, "Llmrouter: An open-source library for llm routing," https://github.com/ulab-uiuc/LLMRouter, 2025, accessed: 2026-03-12.

[4] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, 1954.

[5] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, B. New-Man, B. Yuan, B. Yan, C. Zhang, C. Cosgrove, C. D. Manning, C. Ré, D. Acosta-Navas, D. A. Hudson, E. Zelikman, E. Durmus, F. Ladhak, F. Rong, H. Ren, H. Yao, J. Wang, K. Santhanam, L. Orr, L. Zheng, M. Yuksekgonul, M. Suzgun, N. Kim, N. Guha, N. Chatterji, O. Khattab, P. Henderson, Q. Huang, R. Chi, S. M. Xie, S. Santurkar, S. Ganguli, T. Hashimoto, T. Icard, T. Zhang, V. Chaudhary, W. Wang, X. Li, Y. Mai, Y. Zhang, and Y. Koreeda, "Holistic evaluation of language models," *Transactions on Machine Learning Research*, vol. 2023-August, 2023.

[6] L. Zheng, W. L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging llm-as-a-judge with mt-bench and chatbot arena," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[7] H. Woisetschläger, R. Zhang, S. Wang, and H.-A. Jacobsen, "Mess+: Dynamically learned inference-time llm routing in model zoos with service level guarantees," in *39th Conference on Neural Information Processing Systems (NeurIPS 2025)*, 2025.

[8] J. Yang, Q. Wu, Z. Feng, Z. Zhou, D. Guo, and X. Chen, "Quality-of-service aware llm routing for edge computing with multiple experts," *IEEE Transactions on Mobile Computing*, vol. 24, 2025.

[9] B. Sun, Z. Huang, H. Zhao, W. Xiao, X. Zhang, Y. Li, and W. Lin, "Llumnix: Dynamic scheduling for large language model serving," 2024. [Online]. Available: https://arxiv.org/abs/2406.03243

[10] S. Chang, B. Chen, K. Thangarajah, H. Lutfiyya, and A. E. Hassan, "Adaptive request scheduling for codellm serving with sla guarantees," 2025. [Online]. Available: https://arxiv.org/abs/2506.19677

[11] Z. Jiang, Y. Huang, G. Yu, J. Huang, J. Gu, and M. R. Lyu, "Preserve: Intelligent management for lmaas systems via hierarchical prediction," in *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE)*. Rio de Janeiro, Brazil: IEEE/ACM, 2026.

[12] J. Yang, Q. Wu, Z. Feng, Z. Zhou, D. Guo, and X. Chen, "Quality-of-service aware llm routing for edge computing with multiple experts," 2025. [Online]. Available: https://arxiv.org/abs/2508.00234

[13] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, 2009.

[14] D. M. Manias, I. Shaer, L. Yang, and A. Shami, "Concept drift detection in federated networked systems," in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1–6.

[15] V. Gudepu, V. R. Chintapalli, P. Castoldi, L. Valcarenghi, B. R. Tamma, and K. Kondepu, "The drift handling framework for open radio access networks: An experimental evaluation," *Computer Networks*, vol. 243, p. 110290, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128624001221

[16] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," in *Synthesis Lectures on Communication Networks*, vol. 7, 2010.

[17] D. Ding, A. Mallick, S. Zhang, C. Wang, D. Madrigal, M. D. C. H. Garcia, M. Xia, L. V. S. Lakshmanan, Q. Wu, and V. Rühle, "Best-route: Adaptive llm routing with test-time optimal compute," 2025. [Online]. Available: https://arxiv.org/abs/2506.22716

[18] X. Guo, S. Wang, C. Ji, X. Zhao, W. Xi, Y. Liu, Q. Li, C. Deng, and J. Feng, "Towards generalized routing: Model and agent orchestration for adaptive and efficient inference," *arXiv preprint arXiv:2509.07571*, 2025.