BOUTE: COST-EFFICIENT LLM SERVING WITH HETEROGENEOUS LLMS AND GPUS VIA MULTI-OBJECTIVE BAYESIAN OPTIMIZATION

Youhe Jiang [1] **Fangcheng Fu** [2] **Eiko Yoneki** [1]

ABSTRACT

The rapid growth of large language model (LLM) deployments has made cost-efficient serving systems essential. Recent efforts to enhance system cost-efficiency adopt two main perspectives: (**i**) An *algorithmic* perspective that exploits heterogeneous model capabilities to route simpler queries to lower-cost models and complex queries to higher-cost models (i.e., heterogeneous model routing); and (**ii**) a *systems* perspective that utilizes heterogeneous GPU resources as cost-effective alternatives to homogeneous high-end GPUs (i.e., heterogeneous model deployment). However, algorithm-system co-design for cost-efficient LLM serving necessitates sophisticated management: (**i**) Determining optimal routing strategies under latency and quality requirements, (**ii**) configuring model deployment across heterogeneous GPUs with appropriate resource allocation and parallelism strategies, and (**iii**) co-optimizing routing and deployment decisions to maximize overall system performance. To address these challenges, we present BOUTE, a *quality-aware scheduling system* that jointly exploits heterogeneous model and GPU capabilities for cost-efficient LLM serving. BOUTE employs a *multi-objective Bayesian optimization (MOBO) framework* to co-optimize the routing strategy and model deployment, thereby maximizing the cost-efficiency of the serving system while guaranteeing response quality. Evaluation results demonstrate that BOUTE outperforms state-of-the-art LLM serving systems by up to 157% and 59% on average under *identical* cost budgets and quality requirements, or reducing serving costs by 15%-61% (38% on average) while maintaining the *same* performance targets, validating its effectiveness in achieving cost-efficient LLM serving.

1 INTRODUCTION

Large language models (LLMs) such as Llama-3 (Dubey et al., 2024), DeepSeek-R1 (Guo et al., 2025), Claude (Anthropic, 2025), Gemini (Comanici et al., 2025), and GPT-5 (OpenAI, 2025) have demonstrated outstanding performance across diverse real-world applications, including chatbots, healthcare, and education, profoundly influencing human lives (Jeon & Lee, 2023; Peng et al., 2023; GitHub, 2024). However, LLM serving incurs substantial costs (Jiang et al., 2025a; Miao et al., 2024), as it demands significant computational resources (e.g., GPUs) to satisfy latency and quality requirements. Consequently, how to achieve cost-efficient LLM serving is a timely and essential topic for service providers.

Typically, when a user query arrives, service providers must make two essential decisions during the serving process: (**i**) *which model* should process the query (depending on query complexity and model capability) and (**ii**) *how to deploy* each model across available GPU resources (through resource allocation and parallelism strategies). The ultimate objective of jointly optimizing these two decisions is to achieve cost-efficient LLM serving while maintaining stringent latency and quality requirements. In this work, we explore heterogeneous model routing (Ong et al., 2024; Ding et al., 2024) and heterogeneous model deployment (Jiang et al., 2025b; Mei et al., 2025) to leverage the diverse capabilities of heterogeneous models and GPU resources, thereby enhancing cost-efficiency in LLM serving.

Heterogeneous model routing is a serving paradigm that dynamically assigns incoming queries to the most appropriate model based on predicted response quality requirements. In this approach, a router evaluates each query, and makes a routing decision in a single forward pass (Ong et al., 2024; Ding et al., 2024). Queries deemed easy or low-stakes are directed to smaller, more efficient models, while those identified as difficult or high-stakes are routed to larger, more capable models. By aligning query requirements with model capabilities while satisfying latency and quality requirements, this paradigm enables cost-efficient serving without compromising response quality. Several recent studies (Hu

[1]Department of Computer Science, University of Cambridge, Cambridgeshire, UK [2]School of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, China. Correspondence to: Fangcheng Fu <ccchengff@sjtu.edu.cn>, Eiko Yoneki <eiko.yoneki@cl.cam.ac.uk>.

Proceedings of the 9^{th} MLSys Conference, Bellevue, WA, USA, 2026.

et al., 2024; Chuang et al., 2024; Varangot-Reille et al., 2025; Panda et al., 2025) have focused on cost-efficient LLM serving using heterogeneous model routing.

Recent efforts (Jiang et al., 2025a; Griggs et al., 2024; Mao et al., 2025; Mei et al., 2025) have also proposed leveraging heterogeneous GPU resources for cost-efficient LLM serving (i.e., heterogeneous model deployment). Unlike traditional homogeneous LLM deployments that rely exclusively on high-end GPUs (e.g., H100), this paradigm utilizes a mixture of GPU types, ranging from premium GPUs to more economical alternatives (e.g., RTX 5090), to create a cost-effective serving infrastructure.

As the heterogeneity exists in both models and hardwares, we observe that *heterogeneous model deployment naturally complements heterogeneous model routing*. In particular, in a routing system, since different models exhibit varying resource demands (e.g., compute requirements, memory footprints, and bandwidth utilization), it is intuitive to leverage different types of GPUs tailored to each model's characteristics (Jiang et al., 2025a; Griggs et al., 2024). We present a detailed workload characterization in §3.

Algorithm-system co-design is necessary for heterogeneous model routing (algorithm) and deployment (system). This is due to the bidirectional dependence between these two components: **First**, routing decisions shape the system load distribution (e.g., request arrival rates) across heterogeneous models, directly influencing optimal model deployment strategies. **Conversely**, deployment configurations determine the system latency achievable by each model, which in turn impact routing decisions. Therefore, achieving an efficient, quality-guaranteed serving system necessitates the joint optimization of routing strategy and model deployment. However, realizing such co-optimization in practice proves much harder to implement than to propose:

- **Effective model routing.** Determining an optimal routing strategy requires balancing two competing objectives: system response latency and quality. This decision is further complicated when leveraging heterogeneous resources for serving, where different LLMs on different GPU types exhibit varying system latency, making routing decisions inherently dependent on model deployment.
- **Efficient model deployment.** Different GPU types have distinct characteristics (e.g., compute capability, memory bandwidth, memory capacity), and different LLMs favor different allocations and parallelisms (e.g., data, tensor, pipeline parallelism). This makes model deployment optimization on heterogeneous GPU clusters a complex problem. Co-designing with model routing amplifies this complexity, as routing decisions determine cross-model system loads that influence optimal model deployment.
- **Algorithm-system co-design.** The mutual dependencies between routing and deployment create a challenging circular optimization problem. On the one hand, determining the optimal routing strategy requires knowing the latency and quality characteristics of a given model deployment. On the other hand, selecting the optimal model deployment requires knowing the system load distribution shaped by the routing strategy. This interdependence implies that optimizing routing and deployment in isolation leads to suboptimal system performance.

In order to overcome these challenges, we propose BOUTE, a quality-aware scheduling system that jointly exploits heterogeneous model and GPU capabilities for cost-efficient LLM serving. Our contributions are summarized as follow:

Contribution 1: We systematically characterize the system workload by exploring how co-designing routing and deployment impacts system performance, and demonstrate that heterogeneous model deployment naturally complements heterogeneous model routing: matching the resource demands of different LLMs to appropriate GPU types can substantially improve system efficiency.

Contribution 2: We formulate the scheduling problem of routing and deployment across heterogeneous models and resources as a constraint optimization problem. To solve this problem efficiently, we propose a multi-objective Bayesian optimization (MOBO) framework to co-optimize routing strategy and model deployment. Specifically, given latency and quality objectives, the MOBO framework identifies Pareto-optimal solutions on the latency-quality frontier, enabling service providers to select deployments that best satisfy their performance requirements.

Contribution 3: We implement BOUTE and conduct experiments across diverse LLM tasks and quality requirements. Compared to state-of-the-art LLM serving systems, BOUTE achieves up to 2.6× reduction in system response latency with 1.6× average improvement, or boosts system throughput by up to 1.9× (with 1.6× on average) given the *same* serving budget and quality requirement.

2 BACKGROUND AND RELATED WORKS

Heterogeneous model routing. Model routing systems deploy a router model to dynamically direct incoming queries to different LLMs based on query characteristics, with the primary objective of optimizing system metrics such as cost, latency, or throughput while maintaining output quality. Several recent works (Chen et al.; Hu et al., 2024) have explored this paradigm: RouteLLM (Ong et al., 2024) develops cost-optimal routing strategies using preference data and calibrated confidence scores to balance performance and cost; HybridLLM (Ding et al., 2024) proposes an adaptive LLM routing system that switches between large and small models based on query complexity. These works primarily

focus on router design and routing strategies for query-level model selection, with limited attention to the routing impact on system performance. In contrast, our work, from a service provider perspective, addresses deployment-level optimization to further enhance system performance.

Heterogeneous model deployment. Heterogeneous model deployment refers to LLM serving systems (Jiang et al., 2024; Miao et al., 2024; Mao et al., 2025) that utilize GPUs with varying computational capabilities, memory capacities, or availability characteristics to optimize resource utilization and cost efficiency. Among them, ThunderServe (Jiang et al., 2025b) accommodates the different computational characteristics of prefill and decoding phases by deploying them on heterogeneous GPUs matched to their respective workload requirements; Helix (Mei et al., 2025) formulates heterogeneous clusters as a flow network and utilizes a max-flow algorithm to maximize system throughput. Our work shares a similar motivation: utilizing heterogeneous resources for cost-efficient LLM serving, and we find it naturally complements heterogeneous model routing. A more detailed related work is demonstrated in Appendix D.

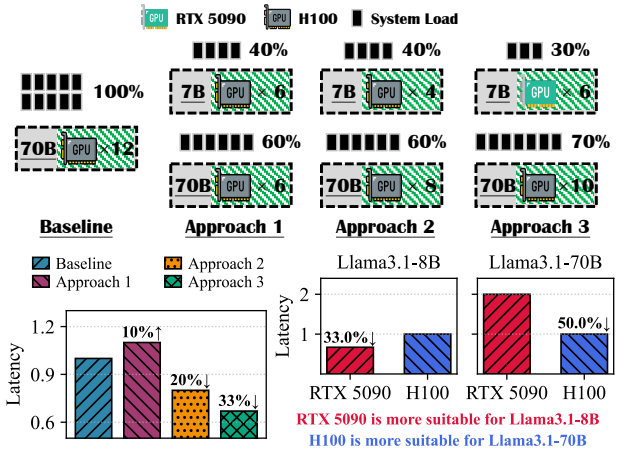

Figure 1. **Upper**: Illustration of system load assignment and resource allocation for approaches 1-3. **Lower Left**: Performance comparison among the baseline and 3 approaches. **Lower Right**: Model performance across GPU types. The latency is normalized by each model's P95 latency on H100 GPUs.

3 WORKLOAD CHARACTERIZATION

This characterization focuses on understanding how heterogeneous model routing and deployment jointly impact system performance and cost efficiency. We introduce different approaches to improve the system performance step-by-step.

Experiment setup. We use P95 latency (the 95th percentile of response latencies) as our primary performance metric and evaluate quality (i.e., accuracy) on a sample of the GSM8K dataset. We compare two deployment configurations under similar budget constraints: a homogeneous setup with 12 H100 GPUs ($32.28/h) and a heterogeneous setup with 6 RTX 5090 GPUs and 10 H100 GPUs ($32.24/h).

Baseline system configuration. We begin by establishing a baseline scenario where a service provider must satisfy a minimum quality requirement of **90** (on a scale of 0-100). Consider a system deploying Llama3.1-70B on 12 H100 GPUs, which achieves a P95 latency of **25.6s** and a quality score of **95.1**. Alternatively, deploying only Llama3.1-8B on the same 12 H100 GPUs yields a P95 latency of **7.4s** but a quality score of **84.5**, failing to meet the required threshold. This illustrates the fundamental trade-off between latency and quality when serving individual models.

Approach 1: Model routing with uniform resource allocation. Introducing model routing enables the system to satisfy quality requirements while reducing latency. With an appropriate routing strategy, approximately 40% of queries are directed to the small model while 60% are routed to the large model, achieving the target quality of **90**. And with uniform resource allocation (6 GPUs per model), the system achieves a P95 latency of **28.2s** (**10%**↑ compared to single Llama3.1-70B model deployment).

Insight: Despite routing reducing the load on the large model, inappropriate resource allocation creates a performance bottleneck. The large model, handling 60% of queries with only half the available resources, becomes the limiting factor for overall system latency.

Approach 2: Optimizing homogeneous resource allocation. The system performance can be substantially improved by adjusting resource allocation. Allocating 4 GPUs to the small model and 8 GPUs to the large model—based on their assigned system loads and model characteristics—reduces the P95 latency to **20.5s** (**20%**↓ compared to single Llama3.1-70B model deployment).

Insight: Model routing combined with appropriate resource allocation significantly reduces system latency. By directing easier queries to the small model and balancing resources between the small and large models to prevent bottlenecks, the system avoids overloading either model.

Approach 3: Introducing heterogeneous GPUs. Further latency reduction is achievable by deploying heterogeneous GPU types matched to model characteristics. In the heterogeneous setup, we route 30% and 70% of queries to the small and large models, respectively, deploying 6 RTX 5090 GPUs for the small model and 10 H100 GPUs for the large model. This configuration achieves a P95 latency of **17.1s** (**33%**↓ and **13%**↓ compared to single Llama3.1-70B and homogeneous model deployment), and a quality score of **91.2** (higher quality). Figure 1 demonstrates the performance

comparison among the baseline and approaches 1-3.

Insight: Integrating heterogeneous GPU resources alters the optimal routing strategy by changing the latency-quality trade-off curves of individual models. The routing distribution shifts from 40/60 to 30/70 for two reasons: First, the small model achieves lower latency at reduced cost on RTX 5090 GPUs; second, this cost saving frees up budget to provision more H100 GPUs for the large model, enabling it to handle increased system loads. Heterogeneous model deployment thus creates a dual benefit—cost-efficient serving of the small model while expanding large model capacity—that jointly minimizes overall system latency.

Model performance across GPU types. To demystify the performance characteristics of different models across GPU types, we conduct additional experiments comparing small and large model deployments on RTX 5090 and H100 GPUs under identical budget constraints. Specifically, we compare deployments using 24 RTX 5090 GPUs ($21.36/h) and 8 H100 GPUs ($21.52/h), representing equivalent costs. Our benchmarking reveals that models exhibit distinct performance profiles on different GPU types, as shown in Figure 1. The small model achieves approximately **1.5**× lower P95 latency on RTX 5090 GPUs compared to H100 GPUs, while the large model achieves **2**× lower P95 latency on H100 GPUs compared to RTX 5090 GPUs.

Insight: Different models favor different GPU types for cost-efficient serving, making heterogeneous GPU deployment essential rather than optional. This hardware heterogeneity naturally complements heterogeneous model routing: routing exploits model diversity to balance latency and quality, while heterogeneous deployment exploits GPU diversity to optimize the cost-performance of each model, creating a synergistic system where both dimensions of heterogeneity work together to maximize overall serving cost-efficiency.

We further validate that this model-GPU affinity generalizes beyond Llama to other popular model families (Qwen3, DeepSeek-distill) and discuss the prevalence of heterogeneous GPU deployments in industrial practice in Appendix E.

Based on these insights, we manage to improve the cost efficiency via heterogeneous model routing and model deployment co-optimization. We first present the problem formulation in §4 and introduce our solution in §5.

4 FORMULATION OF CO-OPTIMIZATION

4.1 Decision Variables in Model Routing

Our router. We adopt a threshold-based routing mechanism consistent with prior works (Ong et al., 2024; Ding et al., 2024). Specifically, for each incoming query, the router computes a routing score that estimates the query's difficulty or the likelihood that a smaller model can produce a satisfactory response. This score is then compared against configurable thresholds $\boldsymbol{\tau} = \{\tau_1, \tau_2, \dots\}$ to make the routing decision. When multiple models are available in the system, queries are routed to model i if their routing score falls within the range $[\tau_{i-1}, \tau_i)$.

Decision variables. The routing thresholds $\boldsymbol{\tau}$ constitute the primary decision variables for the routing strategy. These thresholds control the distribution of queries across models, directly impacting the system's latency-quality trade-off. Higher threshold values route a greater proportion of queries to larger, more capable models, thereby ensuring higher output quality at the cost of increased system latency. Conversely, lower thresholds direct more queries to smaller models, reducing system latency while accepting potential quality degradation. Through systematic optimization of $\boldsymbol{\tau}$, the routing strategy can be adapted to meet specific system objectives and performance constraints.

4.2 Decision Variables in Model Deployment

Deployment configuration. Given a set of available heterogeneous GPUs $\mathbf{D} = \{d_1, d_2, \dots, d_N\}$, where each $d_n \geq 0$ represents the number of GPUs of the n-th type, the deployment problem determines how to allocate these GPUs to serve M model types. The deployment configuration must specify three key aspects: (**i**) Model-GPU assignment: which GPU type(s) should be used to deploy each model, (**ii**) resource allocation: how many GPUs of each type should be allocated to each model, and (**iii**) parallelism strategy: the degree of data, tensor, and pipeline parallelism used to distribute each model across its allocated GPUs.

Our deployment strategy. Let there be M model types. We formulate the deployment problem as finding an optimal allocation matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$, where $a_{m,n}$ is the number of GPUs of type n allocated to model type m, along with parallelism strategies $\mathbf{P} = \{p_1, p_2, \dots, p_M\}$, where p_m specifies the optimal parallelism strategy for model type m. The allocation must satisfy two types of constraints: (**i**) Resource constraints: $\sum_{m=1}^{M} a_{m,n} \leq d_n, \forall n$, i.e., the total allocation does not exceed available resources, and (**ii**) budget constraints: $\sum_{m=1}^{M} \sum_{n=1}^{N} a_{m,n} \cdot b_n \leq B_{\text{cap}}$, where b_n is the cost per unit time for GPU type n and B_{cap} is the total budget, ensuring that the deployment cost remains within the specified budget limit.

Decision variables. The allocation matrix $\mathbf{A}$ and parallelism strategies $\mathbf{P}$ constitute the primary decision variables for the model deployment. Given a routing strategy $\boldsymbol{\tau}$ that determines the system load distribution $\mathbf{W} = \{\lambda_1, \lambda_2, \dots, \lambda_M\}$ among the M model types, where λ_m represents the system load for model type m, we systematically optimize $\mathbf{A}$ and $\mathbf{P}$ to minimize system latency subject to resource and budget constraints.

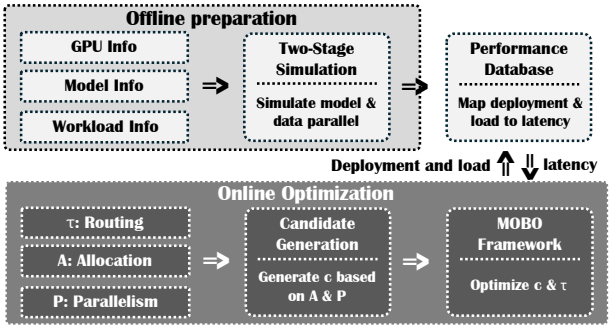

Figure 2. Illustration of the two phases in the MOBO framework.

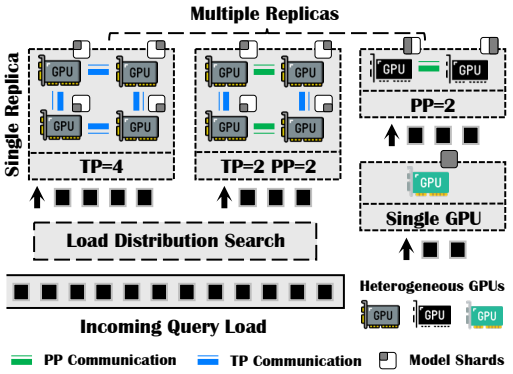

Figure 3. Illustration of single- and multi-replica deployments.

4.3 Problem Formulation

Given a set of M model types and heterogeneous GPUs $\mathbf{D} = \{d_1, d_2, \ldots, d_N\}$ with costs $\mathbf{B} = \{b_1, b_2, \ldots, b_N\}$, we formulate the joint optimization as a multi-objective optimization problem:

$$\min_{\boldsymbol{\tau}, \mathbf{A}, \mathbf{P}} \quad (L(\boldsymbol{\tau}, \mathbf{A}, \mathbf{P}),\ -Q(\boldsymbol{\tau}))$$
$$\text{s.t.} \sum_{m=1}^{M} a_{m,n} \le d_n\ (\forall n), \sum_{m=1}^{M}\sum_{n=1}^{N} a_{m,n} \cdot b_n \le B_{\text{cap}} \tag{1}$$

where $L(\cdot)$ is the system response latency (e.g., P95 latency), $Q(\cdot)$ is the aggregated output quality across all queries (e.g., accuracy on GSM8K), and B_{cap} is the budget constraint. The decision variables are routing thresholds $\boldsymbol{\tau}$, GPU allocation matrix $\mathbf{A}$, and parallelism strategies $\mathbf{P}$. The goal is to find Pareto-optimal solutions that balance the trade-off between minimizing latency and maximizing quality.

This problem is challenging due to the coupling between routing and deployment: routing strategy $\boldsymbol{\tau}$ determines system load distribution across models, which affects required model deployment $(\mathbf{A}, \mathbf{P})$ (i.e., resource allocation and parallelism strategy), while model deployment determines each model's latency characteristics, which influences the optimal routing strategy. Additionally, the problem complexity is further exacerbated by the NP-hardness of heterogeneous resource allocation (Yuan et al., 2022; Jiang et al., 2024). Combined with these factors, the vast search space makes exhaustive search computationally infeasible. Therefore, we design a Multi-Objective Bayesian Optimization (MOBO) framework tailored for efficiently solving this problem.

5 THE MOBO FRAMEWORK

As illustrated in Figure 2, our MOBO framework operates in two phases: (**i**) Offline preparation phase: This phase profiles system performance and simulates different deployment configurations to generate performance data. This preparation is performed *only* once for each specific serving scenario. (**ii**) Online optimization phase: This phase leverages the offline performance data to efficiently search for optimal routing strategies and model deployments.

5.1 Offline Preparation Phase

The offline phase aims to map any ⟨model deployment, system load⟩ pair to an estimated system latency value.

Inference task simulator. An inference task simulator[1] is employed to evaluate deployment performance. We profile heterogeneous GPU characteristics (e.g., effective compute throughput and achievable HBM I/O bandwidth) as inputs to the simulator, and iteratively simulate the performance of different model types under various model deployments $(\mathbf{A}, \mathbf{P})$ across a range of system loads $\lambda = \{2, 4, 6, 8, \ldots, 40\}$ queries per second (QPS, spanning light to burst loads). The resulting performance data is then used in the subsequent online optimization phase.

Two-stage simulation approach. We adopt a two-stage simulation strategy based on parallelism types. **First**, we simulate single-replica deployments using model parallelism, where a model is partitioned across GPUs. This constructs a performance database mapping ⟨single-replica deployment, system load⟩ to system latency. **Second**, we simulate multi-replica deployments using data parallelism (DP) (Li et al., 2023), where multiple replicas serve queries concurrently. This maps ⟨multi-replica deployment, system load⟩ to system latency. We demonstrate the illustration of single- and multi-replica deployments in Figure 3.

Simulation with model parallelism. For each model type and available GPU type, we systematically enumerate feasible deployment configurations using model paral-

[1]We use the ETH EASL Scratchpad simulator (ETH-EASL, 2025) to estimate system P95 latency. We show detailed simulator design (e.g., inputs, batching strategy, queuing mechanism, parallelism strategy modeling) and evaluation in Appendix A.

lelism strategies. Model parallelism includes tensor parallelism (TP) (Shoeybi et al., 2019) and pipeline parallelism (PP) (Huang et al., 2019), which split a single model replica across multiple GPUs. Specifically, for GPU type n with d_n available units, we explore allocations from 1 to d_n GPUs, evaluating all valid TP and PP combinations for each allocation size. The simulator generates the system latency for each specific model parallelism configuration. To constrain the enumeration space, we apply two pruning rules: (**i**) Configurations with insufficient GPU memory for the model are eliminated (e.g., 4 RTX 5090 GPUs for Llama3.1-70B), and (**ii**) TP is restricted to utilize fast intra-machine GPU connections to avoid communication bottlenecks (e.g., TP ≤ 8 or 4, depending on hardware). This enumeration process is repeated across all model types, GPU types, and system loads, generating a performance database mapping model-parallel configurations to their corresponding system latencies. We demonstrate the enumeration complexity in Appendix B.

Simulation with data parallelism. Beyond model parallelism, a model type may be deployed as multiple independent replicas using DP. In this case, determining the optimal load distribution among these DP replicas remains non-trivial, as it directly impacts the final system latency. To address this, we perform an efficient load distribution search: For each multi-replica configuration, we (**i**) enumerate possible load distributions $(\lambda_{m,1}, \lambda_{m,2}, \dots)$ satisfying $\sum_i \lambda_{m,i} = \lambda_m$, where $\lambda_{m,i}$ denotes the system load assigned to replica i of model type m, (**ii**) retrieve the performance database for each replica's system latency under its assigned load, and (**iii**) select the load distribution that minimizes the maximum system latency across different replicas, and record this minimum latency. Since each evaluation requires only database lookups, this search incurs minimal computational overhead. Through this approach, we can obtain the optimal load distribution and estimate system latency for arbitrary DP configurations.

Summary. The offline preparation phase constructs a comprehensive performance database that enables fast online optimization without real-time profiling or simulation overhead. Additionally, the offline preparation process typically completes within several dozen minutes and needs to be run only *once* for each specific serving scenario.

5.2 Online Optimization Phase

The online phase aims to utilize MOBO to find the Pareto-optimal routing strategies and model deployments.

Deployment candidate generation. We need to enumerate different deployments $(\mathbf{A}, \mathbf{P})$ for each model type. However, the number of possible candidates explodes with the number of GPU types, creating an extremely large search space for MOBO. Therefore, we generate deployment candidates for different model types as follows:

- **Homogeneous deployment candidates.** For each system load and GPU type, we enumerate feasible (DP, TP, PP) combinations and apply *Pareto-skimming* in the (latency, cost) space to generate a set of Pareto-optimal configurations (typically 10-20 data points per GPU type with different latency-cost trade-offs). These configurations serve as homogeneous deployment candidates.

- **Heterogeneous deployment candidates.** We generate heterogeneous candidates by combining homogeneous candidates from different GPU types. Specifically, we enumerate a grid of intermediate budgets $o \in (0, B_{\text{cap}}]$ (e.g., 10-20 points). For each budget o, we solve a *multiple-choice knapsack problem* that selects at most one candidate from each GPU type to minimize system latency subject to the constraint that total cost $\leq o$. During knapsack solving, the system latency for each heterogeneous DP configuration is obtained through the load distribution search described in §5.1. The final set of configurations obtained across all budgets serves as heterogeneous deployment candidates.

Through this approach, we effectively cover the entire deployment space by pruning sub-optimal solutions via Pareto-skimming and knapsack optimization, reducing the candidate set to a manageable size without sacrificing solution quality. We denote the set of deployment candidates for model type m as $\mathcal{C}_m$. The deployment choices are represented as $\mathbf{c} = \{c_1, c_2, \dots, c_M\}$, where $c_m \in \mathcal{C}_m$ for all $m \in \{1, \dots, M\}$.

Combining homogeneous candidates via data parallelism across GPU types is an intentional design choice rather than an inherent limitation; we discuss the rationale in Appendix I.

Overall MOBO workflow. The online optimization phase takes decision variables $\boldsymbol{\theta} = (\boldsymbol{\tau}, \mathbf{c})$ as inputs and outputs the outcomes: response quality $Q(\boldsymbol{\theta})$[2] and system latency $L(\boldsymbol{\theta})$[3]. The MOBO framework primarily comprises two components: a surrogate model—a Gaussian Process (GP)—that provides fast statistical approximations of the objectives and constraints, and an acquisition function that balances exploration and exploitation over the decision space. The optimization workflow proceeds as follows: (**i**) Surrogate model fitting: We fit GP surrogates for $Q(\boldsymbol{\theta})$ and $L(\boldsymbol{\theta})$ using additive kernels and structural information to capture the relationship between outcomes and decision variables. (**ii**) Acquisition function optimization: We construct the constrained qNEHVI (q-Noisy Expected Hypervolume Im-

[2]For different routing strategies $\boldsymbol{\tau}$, we use a sample of the incoming queries (i.e., a subsampled trace) to estimate their response quality $Q(\boldsymbol{\theta})$, following prior works (Ong et al., 2024; Ding et al., 2024; Shafran et al., 2025).

[3]Aggregated P95 system latency across all model types.

provement) acquisition function and optimize it over the decision space to balance exploration and exploitation, selecting the next candidate configuration $\boldsymbol{\theta}^*$ to evaluate. (**iii**) Update and refinement: We evaluate the selected configuration $\boldsymbol{\theta}^*$, update the GP surrogates based on the observed performance, and repeat steps (**ii**) and (**iii**) until convergence. Upon convergence, we obtain a *Pareto-optimal set of solutions*, from which the optimal configuration is selected based on specific performance requirements (e.g., target system latency and response quality).

Additive kernels. To enable the surrogate model to capture both the separate effects and the interaction between routing strategies and deployment candidates, we employ a GP with additive kernels:

$$k(\boldsymbol{\theta}, \boldsymbol{\theta}^*) = k_\tau(\boldsymbol{\tau}, \boldsymbol{\tau}^*) \; + \; k_c(\mathbf{c}, \mathbf{c}^*) \; + \; \phi\, k_\times\big((\boldsymbol{\tau}, \mathbf{c}), (\boldsymbol{\tau}^*, \mathbf{c}^*)\big)$$

where k_τ and k_c are ARD Matérn kernels (Snoek et al., 2012) that model the separable main effects of the routing strategies $\boldsymbol{\tau}$ and the deployment candidates $\mathbf{c}$, respectively. The interaction term $k_\times$ is a product kernel (Duvenaud et al., 2013) that captures the interdependencies between $\boldsymbol{\tau}$ and $\mathbf{c}$. ϕ controls the weight of the interaction term and is learned during training (typically small). This additive kernel structure allows the GP to disentangle main effects from interactions (Kandasamy et al., 2015), yielding improved sample efficiency and interpretability (Gardner et al., 2018).

Structural information. Appropriate structural information is essential for solving complex MOBO problems, as it guides the surrogate model and acquisition function toward feasible and promising regions. We incorporate several types of structural information based on algorithmic properties, system constraints, and empirical observations (S1-5):

S1. Algorithmic reparameterization: Load-fraction encoding. Rather than directly optimizing routing thresholds $\boldsymbol{\tau}$, we reparameterize them as load fractions—the proportion of system load assigned to each model type. This reparameterization provides several advantages: Small changes in raw thresholds can cause large, discontinuous jumps in load allocation due to non-uniform score distributions, whereas load fractions map linearly to system load and latency, stabilizing the optimization procedure and enabling natural enforcement of linear constraints (e.g., $\sum_{m=1}^{M} f_m = 1$, where f_m is the load fraction assigned to model type m).

S2. System constraint: Budget enforcement. We enforce budget feasibility through hard constraints on total deployment cost. For any deployment decision $\mathbf{c}$, the accumulated cost across all model types must satisfy $\sum_{m=1}^{M}\sum_{n=1}^{N} a_{m,n} \cdot b_n \le B_{\text{cap}}$, where b_n is the cost per GPU of type n and B_{cap} is the budget limit. Configurations violating this constraint are excluded from the search space.

S3. System constraint: GPU availability. Similarly, we enforce hard constraints on GPU resource availability. For any deployment decision, the accumulated GPU allocation of each type must satisfy $\sum_{m=1}^{M} a_{m,n} \le d_n$ for all GPU types n, where d_n is the available count of GPU type n. Violations are excluded from the search space.

S4. Output transformation: Scale normalization. Due to the large magnitude differences between latency and quality objectives, we apply objective-specific transformations: system latency $L(\boldsymbol{\theta})$ is transformed to $\log(L(\boldsymbol{\theta}))$ to stabilize large variations, while response quality $Q(\boldsymbol{\theta})$ is transformed to $\log\left(\frac{Q(\boldsymbol{\theta})}{1-Q(\boldsymbol{\theta})}\right)$ (logit transformation) to linearize small variations near boundary values. These transformations improve GP conditioning and optimization stability.

S5. Heterogeneous GPU awareness: Model-GPU preference encoding. As demonstrated in §3, different model types exhibit distinct performance characteristics on different GPU types. We encode this observation as a preference matrix $\mathbf{S} \in [0,1]^{M\times N}$, where $s_{m,n}$ represents the suitability of GPU type n for model type m. This structural information is incorporated into the GP through a preference-weighted kernel:

$$\tilde{k}_c(\mathbf{c}, \mathbf{c}^*) = w(\mathbf{c}) \cdot w(\mathbf{c}^*) \cdot k_c(\mathbf{c}, \mathbf{c}^*)$$

where k_c is the base deployment kernel and $w(\mathbf{c}) = \exp\{\beta \sum_{m=1}^{M}\sum_{n=1}^{N} a_{m,n} \cdot s_{m,n}\}$ is a weight function. Here, $a_{m,n}$ denotes the number of GPUs of type n allocated to model type m in configuration $\mathbf{c}$, and β is a learnable parameter controlling the strength of the preference bias. This exponential weighting increases covariance between configurations using preferred GPU types, thereby biasing the GP toward favorable allocations. This structural bias improves sample efficiency by directing early exploration toward promising configurations while remaining data-overridable when other constraints (e.g., budget, availability) favor non-preferred GPU types.

Constrained qNEHVI. We employ constrained qNEHVI (Daulton et al., 2021) to incorporate specific performance constraints, such as $L(\boldsymbol{\theta}) \le L_{\max}$ for system latency and $Q(\boldsymbol{\theta}) \ge Q_{\min}$ for response quality, where $L_{\max}$ and $Q_{\min}$ denote maximum latency and minimum quality requirements, respectively. We reformulate these constraints as feasibility conditions:

$$g_1(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) - L_{\max} \le 0, \quad g_2(\boldsymbol{\theta}) = Q_{\min} - Q(\boldsymbol{\theta}) \le 0$$

We leverage the joint posterior distribution of the GP surrogates for $L(\boldsymbol{\theta})$ and $Q(\boldsymbol{\theta})$ to evaluate constraint satisfaction. The constrained qNEHVI acquisition function computes expected hypervolume improvement by restricting Monte Carlo integration to feasible samples (Balandat et al., 2020):

$$\text{Acq}(\boldsymbol{\theta}) = \mathbb{E}\left[\text{HVI}(\boldsymbol{\theta}) \cdot \mathbb{I}\left[\bigcap_{i=1}^{2}\{g_i(\boldsymbol{\theta}) \le 0\}\right]\right],$$

where the expectation is taken over samples from the joint posterior of $(L(\boldsymbol{\theta}), Q(\boldsymbol{\theta}))$, and $\mathbb{I}[\cdot]$ is the indicator function

for feasibility. This formulation preserves the correlations between objectives and constraints during sampling, ensuring that the optimizer naturally balances Pareto frontier exploration with constraint satisfaction (Gardner et al., 2014).

Summary. The online phase employs a MOBO framework that efficiently searches the joint space of routing strategies and model deployments. By incorporating additive kernels, structural information (e.g., load-fraction encoding, model-GPU preference encoding), and enforcing hard constraints via constrained qNEHVI, the framework converges to Pareto-optimal solutions that balance system latency and response quality under given constraints and requirements.

6 System Implementation

Overall routine. The overall routine of BOUTE operates as follows: To launch the serving process, the MOBO framework generates the optimal routing strategy and model deployment, which are then used to configure the router (Ong et al., 2024) and instantiate heterogeneous models across heterogeneous GPU resources. During serving, incoming queries are processed by a coordinator (Yao) (detailed in Appendix C), which connects to all models and dispatches queries to appropriate ones based on the guidance of scheduling results; generated responses are subsequently collected and returned to the coordinator.

MOBO optimization kernels. We implement several optimization kernels using GPyTorch (Gardner et al., 2018) for an efficient MOBO framework: (**i**) An ARD Matérn–5/2 kernel (`MaternKernel` with `nu=2.5`) over routing features (k_τ); (**ii**) an ARD Matérn–3/2 kernel (`MaternKernel` with `nu=1.5`) over deployment features (k_c); (**iii**) a low-weight interaction term $k_\times$ constructed via `ProductKernel(k_`τ`, k_c)`; and (**iv**) a custom `PreferenceWeightedKernel` that wraps k_c to inject structural preferences for model-to-GPU assignments. Each kernel is wrapped with a `ScaleKernel` to learn kernel-specific output scales, while ARD lengthscales are learned jointly by maximizing the Gaussian process marginal likelihood. The composite kernel ($k_\tau + \tilde{k}_c + k_\times$) is integrated into BoTorch (Balandat et al., 2020) `SingleTaskGP` models for system latency and response quality, which are combined via `ModelListGP`. Outcome transforms are applied as follows: `Log+Standardize` for system latency and `Logit+Standardize` for response quality.

Practical deployment considerations. BOUTE's framework is designed to be extensible to parallelism strategies beyond TP and PP (e.g., expert parallelism for MoE models, sequence parallelism), with no fundamental limitations; we detail the extensibility mechanism in Appendix H. In production, workloads may evolve after deployment—we categorize distribution shifts by their required adaptation and discuss how BOUTE handles each scenario efficiently in Appendix F. When a rescheduling event is triggered, the system transitions to the updated configuration via rolling updates to ensure zero-downtime service, as described in Appendix G.

7 Evaluation

7.1 Experimental Setup

Environments. Our experiments are conducted on 4 types of GPU servers, each with 8 GPUs: NVIDIA H100-80G ($2.64-3.07/h), RTX PRO 6000-96G ($1.84-2.09/h), RTX 5090-32G ($0.89/h), and RTX 4090-24G ($0.59/h) GPUs. Within H100 servers, the GPUs are connected via NVLink with a bandwidth of 300GB/s; within RTX 5090 and 4090 servers, the GPUs are connected via PCIe with a bandwidth of 60GB/s. We compare homogeneous and heterogeneous model deployments under the same price budget of $30/h.

Models, router, and traces. We employ Llama3.1-8B (smaller) and Llama3.1-70B (larger) as two model types in our system, which are representative and popular open-source transformer models (Dubey et al., 2024). For model routing, we adopt the router from (Ong et al., 2024), which is trained on the Chatbot Arena dataset augmented with GPT-4-as-a-judge labeled data and golden-labeled MMLU validation data. The routing overhead remains negligible, accounting for less than 1% of the total inference cost. And we follow prior work to generate workload traces based on real-world data (Peng et al., 2025; Zhong et al., 2024). Our testing traces are subsampled from GSM8K (Cobbe et al., 2021) and MTBench (Zheng et al., 2023).

Evaluation metrics. Following previous evaluation setups (Miao et al., 2024), we evaluate system performance based on overall throughput and various percentile latencies (i.e., average, P95, . . . , P99, P100 latencies). In particular, the P95 latency denotes the maximum response time within which 95% of all requests are completed.

For MT-Bench evaluation, we use LLM-as-a-judge with GPT-4 as the judge to score responses from Llama-3.1 models, following standard practice in LLM routing research (Ong et al., 2024; Chen et al.).

Baselines. We compare BOUTE against four baseline systems: (**i**) **Stand-Alone-Homo**: vLLM (Kwon et al., 2023) serving the Llama3.1-70B model with homogeneous resources (H100 GPUs); (**ii**) **Stand-Alone-Hetero**: vLLM serving the Llama3.1-70B model with heterogeneous resources; (**iii**) **RouteLLM**: RouteLLM (Ong et al., 2024), a state-of-the-art open-source framework for LLM routing, routing queries between Llama3.1-8B and Llama3.1-70B models based on query complexity, where each model is served by vLLM with uniform resource allocation across model types; (**iv**) **BOUTE-Homo**: BOUTE serving

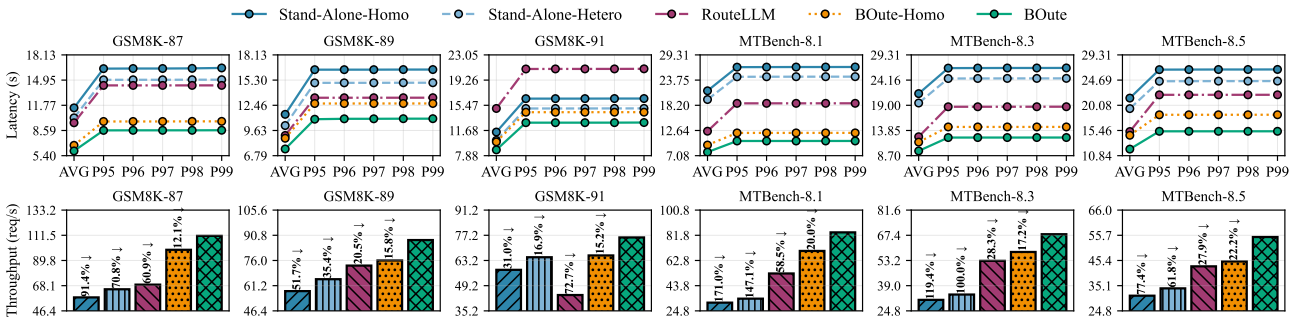

Figure 4. Experimental results of BOUTE compared with different baselines on GSM8K and MTBench workloads under different quality requirements. We select the minimum P95 latency routing strategy and model deployment from the Pareto-optimal solution set. GSM8K-87 represents a quality requirement of 87 (in terms of aggregated accuracy), MTBench-8.1 represents a quality requirement of 8.1 (in terms of aggregated score), and so on.

Llama3.1-8B and Llama3.1-70B models with routing and deployment optimization on homogeneous resources. We select RouteLLM with uniform resource allocation to reflect current industry practice; we discuss the baseline selection rationale in detail in Appendix L. We compare the system performance of BOUTE with these baselines under quality requirements of 86, 88, and 90 for GSM8K and 8.1, 8.3, and 8.5 for MTBench. The system load is set to 100 req/s in total. For fair comparison, we tune the model deployment for each baseline under price budgets and report the best values in all experiments.

7.2 End-to-end Evaluation

End-to-end system performance on GSM8K workloads. As demonstrated in Figure 4 (left), we evaluate BOUTE with heterogeneous model deployment against baseline methods to demonstrate its effectiveness in achieving cost-efficient LLM serving on the GSM8K workload:

(**i**) Compared to standalone LLM serving using Llama3.1-70B with vLLM (Homo and Hetero), BOUTE achieves superior performance across all quality requirements, reducing P95 latency by up to 91% (57% on average) and improving throughput by up to 90% (55% on average). Standalone LLM serving fails to leverage heterogeneous model resources, naively routing queries of varying complexity to a single model type, which results in either poor relative quality or excessive P95 latency. (**ii**) Compared to RouteLLM, BOUTE reduces P95 latency by up to 66% (50% on average) and improves throughput by up to 71% (52% on average). RouteLLM with uniform resource allocation across model types fails to account for the distinct characteristics of different model types, as larger models typically require more computing resources even when assigned fewer workloads. For instance, on the GSM8K trace with a quality requirement of 91, RouteLLM achieves even lower throughput than standalone deployment, as 70% of the queries are processed by the large model with uniform resource allocation. (as detailed in §3). (**iii**) Compared to BOUTE-Homo, BOUTE reduces P95 latency by up to 15% (14% on average) and improves throughput by up to 16% (14% on average). Although BOUTE-Homo incorporates adaptive resource allocation based on system load, it fails to fully exploit the GPU preferences of different model types (as discussed in §3), thereby limiting cost-efficiency.

In contrast to these approaches, BOUTE jointly optimizes both the routing strategy (determining which model serves which query) and the deployment strategy (determining which GPU serves which model), thereby achieving superior cost-efficiency in LLM serving.

End-to-end system performance on MTBench workloads. We conduct additional experiments to evaluate the performance of BOUTE on MTBench workloads with varying quality requirements. As shown in Figure 4 (right), BOUTE achieves up to 157% (115% on average) performance improvement compared to standalone LLM serving (Homo and Hetero), up to 80% (42% on average) performance improvement compared to RouteLLM, and up to 21% (20% on average) performance improvement compared to BOUTE-Homo. These results demonstrate the comprehensive cost-efficiency of BOUTE across diverse workloads.

7.3 Scheduling Evaluation

Scheduling results. Table 1 presents the scheduling outcomes of our MOBO framework, yielding several notable insights into routing and resource allocation strategies: (**i**) Load distribution adapts to quality requirements. As quality requirements increase, the framework progressively shifts system load and computational budget toward the Llama3.1-70B model. For example, when evaluated on the GSM8K trace with a quality threshold of 87, the small and large

Table 1. Routing strategies and resource allocations of BOUTE used in Figure 4.

Trace-Quality	Model Type	Load	Budget	Allocated Resources
GSM8K-87	Llama3.1-8B	73.45%	38.58%	8×5090+5×4090
	Llama3.1-70B	26.55%	61.42%	6×H100
GSM8K-89	Llama3.1-8B	43.95%	17.86%	6×5090
	Llama3.1-70B	56.05%	82.14%	8×H100
GSM8K-91	Llama3.1-8B	29.94%	9.28%	3×5090
	Llama3.1-70B	70.06%	90.72%	6×H100+4×6000
MTBench-8.1	Llama3.1-8B	80.58%	46.34%	8×5090+8×4090+6000
	Llama3.1-70B	19.42%	53.66%	4×H100+2×6000
MTBench-8.3	Llama3.1-8B	61.17%	24.44%	8×5090
	Llama3.1-70B	38.83%	75.56%	6×H100+2×6000
MTBench-8.5	Llama3.1-8B	41.75%	12.04%	4×5090
	Llama3.1-70B	58.25%	87.96%	6×H100+4×6000

Table 2. The scheduling (algorithm converge) time and scalability of BOUTE. BOUTE (w/o structural info) disables algorithmic reparameterization, output transformation, and heterogeneous GPU awareness. BOUTE (w/o offline prep) disables the performance database. The algorithm is considered converged when the solution remains stable for more than 20 consecutive iterations.

GPU Types / GPU Number	BOUTE	BOUTE (w/o structural info)	BOUTE (w/o offline prep)
3 / 24	23.5 s	3.6 min	≈ 12 min
4 / 32	32.5 s	5.3 min	≈ 20 min
5 / 40	48.6 s	7.1 min	> 30 min
6 / 48	1.2 min	8.9 min	> 30 min

models handle 73% and 27% of the system load, respectively. However, when the quality requirement is raised to 91, this distribution shifts to 30% and 70%, reflecting the increased reliance on the more capable large model to meet stringent quality constraints. (**ii**) Resource/Budget allocation reflects model-specific resource requirements. The framework allocates computational resources based not only on assigned system load but also on intrinsic model characteristics (e.g., compute and memory demands). For instance, when tested on MTBench with a quality requirement of 8.3, the small and large models are assigned 61% and 39% of the system load, respectively. However, only 24% of the total budget is allocated to the small model, as it requires substantially fewer computational and memory resources to achieve comparable performance. To prevent the large model from becoming a system bottleneck, the framework allocates 76% of the budget to it, ensuring balanced system performance across both model types. (**iii**) Heterogeneous GPU allocation optimizes resource utilization. The framework strategically assigns heterogeneous GPU resources to different model types based on their computational requirements and GPU availability. For example, when tested on MTBench with a quality requirement of 8.1, all RTX 5090 and 4090 GPUs are allocated to the small model, supplemented by one RTX PRO 6000 GPU due to availability constraints. Conversely, the large model are deployed on four H100 GPUs and two RTX PRO 6000 GPUs, which offer superior compute and memory capabilities aligned with the model's higher resource demands. These results verify that BOUTE is able to take into account the heterogeneity of models and hardware, quality requirements, as well as budget and GPU constraints, generating routing strategies and model deployments that maximize system performance.

Scheduling time. We demonstrate the scheduling time of BOUTE across different cluster scales. As shown in Table 2, BOUTE requires less than 30 seconds to converge to the Pareto-optimal solutions when scheduling on 3 GPU types with 24 total GPUs. As the number of GPU types and total GPUs increases, the scheduling time increases near-linearly, demonstrating its strong scalability across different cluster scales. To further validate the effectiveness of injecting structural information, we evaluate BOUTE (w/o structural info) that disables the structural information described in §5.2. Our experiments show that without structural information, BOUTE requires approximately 8× more BO evaluations to converge. Consequently, the overall scheduling time increases from under a minute to 4-9 minutes for 3-6 GPU types (24-48 total GPUs). Finally, we evaluate BOUTE (w/o offline prep) to validate the effectiveness of our offline preparation phase. Without offline preparation, the scheduling time increases dramatically to 12-30+ minutes for 3-6 GPU types (24-48 total GPUs). This substantial increase occurs because each Bayesian optimization evaluation must re-run the simulator to obtain performance metrics for different candidate deployments, making the optimization process significantly more expensive.

We further conduct ablation experiments to isolate the contribution of each structural component (S1, S4, S5) individually; see Appendix J for detailed results.

7.4 Case Study

Pareto optimality of BOUTE. To further validate our approach, we evaluate BOUTE across varying quality requirements and compare its P95 latency against BOUTE-Homo, BOUTE (w/o structural info), and RouteLLM. As illustrated in Figure 5, BOUTE consistently achieves superior P95 latency performance across different quality requirements. Specifically, we observe the following: (**i**) BOUTE exhibits Pareto dominance over BOUTE-Homo through its integration of heterogeneous resources and effective scheduling. Under equivalent quality requirements, BOUTE achieves lower latency; conversely, for a given latency constraint, BOUTE delivers higher quality. (**ii**) BOUTE similarly demonstrates Pareto dominance over BOUTE (w/o structural info). Despite the additional scheduling overhead mentioned in §7.3, BOUTE without structural information frequently converges to local optima, resulting in inferior performance. In most cases, BOUTE (w/o structural info) is worse than not only BOUTE but also BOUTE-Homo. (**iii**) RouteLLM with uniform resource allocation demonstrate

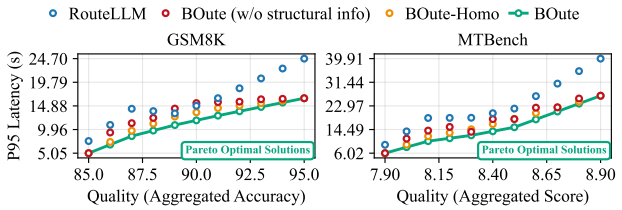

Figure 5. P95 latency results across different quality requirements.

Table 3. Costs required to meet quality and latency requirements. (87, 8) represents quality and latency requirements of 87 and 8s.

Baseline	GSM8K (87, 8)	GSM8K (91, 12)	MTBench (8.1, 10)	MTBench (8.5, 15)
Stand-Alone-Homo	$61.83/h	$41.21/h	$79.93/h	$53.29/h
Stand-Alone-Hetero	$56.24/h	$37.50/h	$73.54/h	$49.02/h
RouteLLM	$53.55/h	$52.38/h	$55.90/h	$44.00/h
BOUTE-Homo	$38.41/h	$36.07/h	$36.33/h	$38.31/h
BOUTE	**$32.25/h**	**$32.18/h**	**$30.98/h**	**$31.23/h**

the worst performance across different quality requirements. These results collectively (**i**) demonstrate the efficacy of integrating heterogeneous resources, (**ii**) underscore the critical importance of incorporating structural information into the scheduling process, and (**iii**) highlight the importance of resource allocation optimization in a routing system.

Cost-efficiency of BOUTE. We evaluate the cost-efficiency of BOUTE under specified latency and quality requirements. Under equivalent quality constraints, systems typically require increased GPU resource allocation (higher budget) to achieve lower latency. Table 3 presents comparative results across multiple benchmark traces under varying quality and P95 latency requirements. BOUTE achieves the same latency and quality targets while reducing costs by an average of 39.7% compared to Stand-Alone baselines and 38.0% compared to RouteLLM. Moreover, BOUTE demonstrates a 15.0% cost reduction relative to BOUTE-Homo, underscoring the advantages of heterogeneous model deployment. These results validate the effectiveness of BOUTE in reducing operational costs for real-time LLM serving systems.

Quality signal and deployment reality. BOUTE's optimization framework is agnostic to the specific quality metric employed; we discuss practical quality proxies, their limitations, and alignment with deployment reality in Appendix K.

8 CONCLUSION

This work focuses on cost-efficient LLM serving with heterogeneous models and GPUs. It proposes BOUTE, a serving system that employs a multi-objective Bayesian optimization (MOBO) framework to jointly optimize quality and latency through adaptive routing decisions for different queries and strategic model deployment across diverse models and hardware. Results show that, compared with baselines, BOUTE achieves up to 2.57× lower latency (1.58× on average) under identical cost budgets and quality requirements, or alternatively reduces costs by 15%-61% while maintaining the same performance targets. These results demonstrate BOUTE's cost-efficiency, and we believe it can contribute to the democratization of LLM serving.

REFERENCES

Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B., Tumanov, A., and Ramjee, R. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), pp. 117–134, 2024.

Anthropic. Claude: Family of language models, 2025. URL https://www.anthropic.com/.

Balandat, M., Karrer, B., Jiang, D., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. Botorch: A framework for efficient monte-carlo bayesian optimization. Advances in neural information processing systems, 33:21524–21538, 2020.

Chen, L., Zaharia, M., and Zou, J. Frugalgpt: How to use large language models while reducing cost and improving performance. Transactions on Machine Learning Research.

Chuang, Y.-N., Sarma, P. K., Gopalan, P., Boccio, J., Bolouki, S., Hu, X., and Zhou, H. Learning to route llms with confidence tokens. arXiv preprint arXiv:2410.13284, 2024.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

Comanici, G., Bieber, E., Schaekermann, M., Pasupat, I., Sachdeva, N., Dhillon, I., Blistein, M., Ram, O., Zhang, D., Rosen, E., et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. arXiv preprint arXiv:2507.06261, 2025.

Daulton, S., Balandat, M., and Bakshy, E. Parallel bayesian optimization of multiple noisy objectives with expected hypervolume improvement. Advances in neural information processing systems, 34:2187–2200, 2021.

Ding, D., Mallick, A., Wang, C., Sim, R., Mukherjee, S., Ruhle, V., Lakshmanan, L. V., and Awadallah, A. H. Hybrid llm: Cost-efficient and quality-aware query routing. arXiv preprint arXiv:2404.14618, 2024.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.

Duvenaud, D., Lloyd, J., Grosse, R., Tenenbaum, J., and Zoubin, G. Structure discovery in nonparametric regression through compositional kernel search. In International Conference on Machine Learning, pp. 1166–1174. PMLR, 2013.

ETH-EASL. Scratchpad, 2025. URL https://github.com/eth-easl/Scratchpad.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. Advances in neural information processing systems, 31, 2018.

Gardner, J. R., Kusner, M. J., Xu, Z. E., Weinberger, K. Q., and Cunningham, J. P. Bayesian optimization with inequality constraints. In ICML, volume 2014, pp. 937–945, 2014.

GitHub. The world's most widely adopted ai developer tool, 2024. URL https://github.com/features/copilot.

Griggs, T., Liu, X., Yu, J., Kim, D., Chiang, W.-L., Cheung, A., and Stoica, I. M\'elange: Cost efficient large language model serving by exploiting gpu heterogeneity. arXiv preprint arXiv:2404.14527, 2024.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948, 2025.

Hu, Q. J., Bieker, J., Li, X., Jiang, N., Keigwin, B., Ranganath, G., Keutzer, K., and Upadhyay, S. K. Routerbench: A benchmark for multi-llm routing system. arXiv preprint arXiv:2403.12031, 2024.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. Advances in neural information processing systems, 32, 2019.

Jeon, J. and Lee, S. Large language models in education: A focus on the complementary relationship between human teachers and chatgpt. Education and Information Technologies, 28(12):15873–15892, 2023.

Jiang, Y., Yan, R., Yao, X., Zhou, Y., Chen, B., and Yuan, B. Hexgen: generative inference of large language model over heterogeneous environment. In Proceedings of the 41st International Conference on Machine Learning, pp. 21946–21961, 2024.

Jiang, Y., Fu, F., Yao, X., He, G., Miao, X., Klimovic, A., Cui, B., Yuan, B., and Yoneki, E. Demystifying cost-efficiency in llm serving over heterogeneous gpus. arXiv preprint arXiv:2502.00722, 2025a.

Jiang, Y., Fu, F., Yao, X., Wang, T., Cui, B., Klimovic, A., and Yoneki, E. Thunderserve: High-performance and cost-efficient llm serving in cloud environments. arXiv preprint arXiv:2502.09334, 2025b.

Jiang, Y., Yan, R., and Yuan, B. Hexgen-2: Disaggregated generative inference of llms in heterogeneous environment. arXiv preprint arXiv:2502.07903, 2025c.

Kandasamy, K., Schneider, J., and Póczos, B. High dimensional bayesian optimisation and bandits via additive models. In International conference on machine learning, pp. 295–304. PMLR, 2015.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In Proceedings of the 29th symposium on operating systems principles, pp. 611–626, 2023.

Li, Z., Zheng, L., Zhong, Y., Liu, V., Sheng, Y., Jin, X., Huang, Y., Chen, Z., Zhang, H., Gonzalez, J. E., et al. {AlpaServe}: Statistical multiplexing with model parallelism for deep learning serving. In 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23), pp. 663–679, 2023.

LibP2P. A modular network stack, 2023. URL https://libp2p.io/.

Mao, Z., Xia, T., Wu, Z., Chiang, W.-L., Griggs, T., Bhardwaj, R., Yang, Z., Shenker, S., and Stoica, I. Skyserve: Serving ai models across regions and clouds with spot instances. In Proceedings of the Twentieth European Conference on Computer Systems, pp. 159–175, 2025.

Mei, Y., Zhuang, Y., Miao, X., Yang, J., Jia, Z., and Vinayak, R. Helix: Serving large language models over heterogeneous gpus and network via max-flow. In Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, pp. 586–602, 2025.

Miao, X., Shi, C., Duan, J., Xi, X., Lin, D., Cui, B., and Jia, Z. Spotserve: Serving generative large language models on preemptible instances. In Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, pp. 1112–1127, 2024.

Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., Gonzalez, J. E., Kadous, M. W., and Stoica, I. Routellm: Learning to route llms with preference data. arXiv preprint arXiv:2406.18665, 2024.

OpenAI. Gpt-5 system card, 2025. URL https://cdn.openai.com/gpt-5-system-card.pdf.

Panda, P., Magazine, R., Devaguptapu, C., Takemori, S., and Sharma, V. Adaptive llm routing under budget constraints. arXiv preprint arXiv:2508.21141, 2025.

Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, Í., Maleki, S., and Bianchini, R. Splitwise: Efficient generative llm inference using phase splitting. In 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA), pp. 118–132. IEEE, 2024.

Peng, C., Yang, X., Chen, A., Smith, K. E., PourNejatian, N., Costa, A. B., Martin, C., Flores, M. G., Zhang, Y., Magoc, T., et al. A study of generative large language model for medical research and healthcare. NPJ digital medicine, 6(1):210, 2023.

Peng, Y., Jiang, Y., Wang, C., and Yuan, B. Hexgen-text2sql: Optimizing llm inference request scheduling for agentic text-to-sql workflow. arXiv preprint arXiv:2505.05286, 2025.

Shafran, A., Schuster, R., Ristenpart, T., and Shmatikov, V. Rerouting llm routers. arXiv preprint arXiv:2501.01818, 2025.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053, 2019.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems, 25, 2012.

Varangot-Reille, C., Bouvard, C., Gourru, A., Ciancone, M., Schaeffer, M., and Jacquenet, F. Doing more with less–implementing routing strategies in large language model-based systems: An extended survey. arXiv preprint arXiv:2502.00409, 2025.

Wu, B., Zhong, Y., Zhang, Z., Liu, S., Liu, F., Sun, Y., Huang, G., Liu, X., and Jin, X. Fast distributed inference serving for large language models. arXiv preprint arXiv:2305.05920, 2023.

Yao, X. Open compute framework: Peer-to-peer task queue for foundation model inference serving, september 2023. URL https://github.com/autoai-org/OpenComputeFramework.

Yuan, B., He, Y., Davis, J., Zhang, T., Dao, T., Chen, B., Liang, P. S., Re, C., and Zhang, C. Decentralized training of foundation models in heterogeneous environments. Advances in Neural Information Processing Systems, 35: 25464–25477, 2022.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. Advances in Neural Information Processing Systems, 36: 46595–46623, 2023.

Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., and Zhang, H. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), pp. 193–210, 2024.

Table 4. Simulator accuracy across parallelism configurations on Llama3-70B model under a workload with average input and output lengths of 1600 and 16. Errors are absolute percentage errors.

Config (DP,TP,PP)	Real (req/s)	Estimated (req/s)	Abs. % Error
(1, 4, 1)	0.21	0.219	4.29%
(1, 2, 2)	0.26	0.280	7.69%
(1, 1, 4)	0.27	0.287	6.30%
(2, 1, 2)	0.33	0.347	5.15%
(2, 2, 1)	0.40	0.408	2.00%
(2, 4, 1)	0.41	0.437	6.59%
(2, 2, 2)	0.55	0.559	1.64%

A Simulator Design and Validation

Our simulator employs a round-robin strategy for request dispatching among multiple parallel models, and a first-come first-served strategy for per-model request processing. The single-GPU processing time is based on profiled characteristics like compute TFLOPS and memory bandwidth. The simulator also considers the phase-specific characteristics of LLMs. The prefill phase is compute-bound, so its batched processing capacity is determined by the sum of the individual latencies. In contrast, the decoding phase is memory-bound, and its batched processing capability is defined by a single latency value. This distinction has been validated in several studies (e.g., DistServe (Zhong et al., 2024), Splitwise (Patel et al., 2024)).

Simulator inputs. In addition to GPU characteristics, the simulator requires a subsampled LLM workload trace. The workload trace is essential for accurate performance modeling, as it provides workload-specific characteristics such as input and output sequence length distributions that directly impact system performance.

Batching strategy in our simulator. The simulator's internal batching strategy is continuous batching, which iteratively batches request tokens to fully utilize the current resources. The GPU's memory limit constrains the maximum batch size for continuous batching.

Queuing mechanism. Our simulator maintains an individual queue for each model. Once there is free memory on the GPU (one request has finished), the model will fetch the next request in the queue for processing.

Different parallelism. Tensor and pipeline parallelism both split the computation workload of a single model across multiple devices. For pipeline parallelism, the simulator models communication overhead by profiling the relationship between estimated communication volume and observed latency. For tensor parallelism, the simulator assumes that each operator's computation cost ideally scales down by a factor of $1/N$ when split across N GPUs, and then adjusts this ideal cost using a speed-up coefficient $K(N)$ obtained from micro-benchmarks to account for communication and synchronization overhead. All profiling is performed offline before scheduling begins.

Simulator evaluation. We present the accuracy of our simulator with real-time experiments in Table 4. The table presents examples of our throughput estimation for the Llama3-70B model under a workload with average input and output lengths of 1600 and 16, respectively. The notation (1,2,2) indicates a DP degree of 1, TP degree of 2, and PP degree of 2. Although the estimations are not perfectly accurate, they are sufficiently reliable (with estimation errors within 2%–7%) for selecting optimal configurations.

B Complexity Analysis of First-Stage Offline Simulation

For each model type and available GPU type, we systematically enumerate feasible deployment configurations using model parallelism strategies. Model parallelism includes tensor parallelism (TP) (Shoeybi et al., 2019) and pipeline parallelism (PP) (Huang et al., 2019), which split a single model replica across multiple GPUs. Specifically, for GPU type $n \in \{1, \ldots, N\}$ with d_n available units, we explore allocations from 1 to d_n GPUs, evaluating all valid TP and PP combinations for each allocation size. For example, an 8-GPU allocation is simulated under configurations (TP=1, PP=8), (TP=2, PP=4), (TP=4, PP=2), and (TP=8, PP=1). The enumeration complexity is $\Theta\left(|\lambda| M \sum_{n=1}^{N} d_n \log d_n\right)$, where $|\lambda|$ represents the number of enumerated system load values, and the logarithmic factor accounts for valid TP and PP combinations per GPU allocation size. After pruning, the enumeration complexity can be reduced to $\Theta\left(|\lambda| M \sum_{n=1}^{N} d_n\right)$.

C Extended System Implementation

Overall routine. The overall routine of BOUTE operates as follows: To launch the serving process, the MOBO framework generates the optimal routing strategy and model deployment, which are then used to configure the router and instantiate heterogeneous models across heterogeneous GPU resources. During serving, incoming queries are processed by a coordinator, which connects to all models and dispatches queries to appropriate ones based on the guidance of scheduling results; generated responses are subsequently collected and returned to the coordinator.

Inference task coordinator. To deploy BOUTE in a heterogeneous environment, we employ a task coordinator that incorporates a router responsible for generating routing scores for input queries (Ong et al., 2024; Ding et al., 2024) and dispatches queries based on both these scores and the optimal routing strategy obtained from our scheduling algorithm. The task coordinator is primarily built upon an open-source implementation of decentralized computa-

tion coordination (Yao) that utilizes libP2P (LibP2P, 2023) to establish peer-to-peer network connections between the coordinator and distributed models. When the task coordinator receives an input query, the routing process proceeds in three steps: **First**, the router evaluates the query to determine the appropriate model type based on its routing score. **Second**, the coordinator selects a specific replica of the chosen model type by consulting the local distribution search results described in §5.1, which balance workloads across replicas according to their heterogeneous deployment capabilities. **Third**, the coordinator directs the query to the selected replica for execution and response generation.

MOBO optimization kernels. We implement several optimization kernels using GPyTorch (Gardner et al., 2018) for an efficient MOBO framework: (**i**) an ARD Matérn–5/2 kernel (`MaternKernel` with `nu=2.5`) over routing features (k_τ) to capture smooth, continuous effects of threshold variations; (**ii**) an ARD Matérn–3/2 kernel (`MaternKernel` with `nu=1.5`) over deployment features (k_c) to accommodate rougher, step-like transitions when switching between deployment candidates; (**iii**) a low-weight interaction term $k_\times$ constructed via `ProductKernel(k_τ, k_c)` to capture cross-effects between routing and deployment features; and (**iv**) a custom `PreferenceWeightedKernel` that wraps k_c to inject structural preferences for model-to-GPU assignments, yielding a preference-weighted deployment kernel $\tilde{k}_c$. Each kernel component is wrapped with a `ScaleKernel` to learn component-specific output scales, while ARD lengthscales are learned jointly by maximizing the Gaussian process marginal likelihood. The composite kernel ($k_\tau + \tilde{k}_c + k_\times$) is integrated into BoTorch (Balandat et al., 2020) `SingleTaskGP` models for system latency and response quality, which are combined via `ModelListGP`. Outcome transforms are applied as follows: `Log+Standardize` for system latency and `Logit+Standardize` for response quality. All kernels are executed in double precision.

D Extended Related Works

LLM serving systems. Recent works have proposed several systems and methods for highly efficient LLM serving. AlpaServe (Li et al., 2023) optimizes system service level objectives (SLOs) by utilizing data and model parallelism; DistServe (Zhong et al., 2024) and Splitwise (Patel et al., 2024) split the prefill and decoding phases onto separate GPUs to eliminate interference between them; SarathiServe (Agrawal et al., 2024) optimizes request batching through prefill chunking to mitigate interference between the prefill and decoding stages; vLLM (Kwon et al., 2023) introduces PagedAttention for efficient memory management, enabling higher batch sizes and improved throughput; FastServe (Wu et al., 2023) employs a preemptive scheduling mechanism that prioritizes shorter jobs to minimize job completion time. These works primarily focus on improving serving performance through techniques such as batching optimization, phase separation, and memory management. In contrast, our work focuses on deployment scheduling with heterogeneous LLMs and GPU types to achieve cost-efficient LLM serving.

Heterogeneous model routing. Model routing systems deploy a router model to dynamically direct incoming queries to different LLMs based on query characteristics, with the primary objective of optimizing system metrics such as cost, latency, or throughput while maintaining output quality. Several recent works have explored this paradigm: RouteLLM (Ong et al., 2024) develops cost-optimal routing strategies using preference data and calibrated confidence scores to balance performance and cost; HybridLLM (Ding et al., 2024) proposes an adaptive LLM routing system that switches between large and small models based on query complexity; FrugalGPT (Chen et al.) cascades LLMs from weakest to strongest, stopping when a model produces a satisfactory response to minimize costs; Router-Bench (Hu et al., 2024) provides a comprehensive benchmark for evaluating routing strategies across diverse query types and model combinations. These works primarily focus on router design and routing strategies for query-level model selection, with limited attention to the routing impact on system performance. In contrast, our work, from a service provider perspective, addresses deployment-level optimization to further enhance system performance.

Heterogeneous model deployment. Heterogeneous model deployment refers to LLM serving systems that utilize GPUs with varying computational capabilities, memory capacities, or availability characteristics to optimize resource utilization and cost efficiency. Among them, HexGen (Jiang et al., 2025c) proposes asymmetric model parallelism and evolutionary-based scheduling for distributing computation and communication among heterogeneous GPUs and networks; ThunderServe (Jiang et al., 2025b) accommodates the different computational characteristics of prefill and decoding phases by deploying them on heterogeneous GPUs matched to their respective workload requirements; Helix (Mei et al., 2025) formulates heterogeneous clusters as a flow network and utilizes a max-flow algorithm to maximize system throughput; SpotServe (Miao et al., 2024) suggests utilizing spot instances for cheap and cost-efficient LLM serving; SkyServe (Mao et al., 2025) provides a multi-cloud serving framework that automatically provisions LLM replicas across different cloud providers and regions to optimize cost and availability. Our work shares a similar motivation: utilizing heterogeneous resources for cost-efficient LLM serving, and we find it naturally complements heterogeneous model routing.

E GENERALIZATION AND INDUSTRIAL PREVALENCE

E.1 Generalization to Other Model Families

To validate that the model-GPU affinity observed in Figure 1 generalizes beyond Llama, we conduct additional experiments on Qwen3 and DeepSeek-distill model families following the same setup as Figure 1.

Table 5. Per-cost P95 latency (s) across model families and GPU types. The percentage indicates the latency reduction achieved by the more cost-efficient GPU type.

Model	RTX 5090	H100
Qwen3-8B	**20.52s** (41.9%↓)	35.30s
Qwen3-32B	184.27s	**132.73s** (28.0%↓)
DeepSeek-distill-8B	**17.43s** (50.0%↓)	34.88s
DeepSeek-distill-70B	461.38s	**292.04s** (36.7%↓)

As shown in Table 5, smaller models (8B) consistently achieve lower per-cost latency on RTX 5090 GPUs, while larger models (32B–70B) favor H100 GPUs. This confirms that the model-GPU affinity exploited by BOUTE is a general phenomenon across popular open-source model families, not an artifact specific to Llama.

E.2 Prevalence of Heterogeneous GPU Deployments in Industry

Large-scale serving systems rarely operate on a single generation of hardware. Due to rapid GPU release cycles (e.g., V100 → A100 → H100 → B100) and supply chain constraints, datacenters inevitably accumulate a mix of GPU generations.

Google Cloud. As noted in Helix (Mei et al., 2025), Google Cloud datacenters are equipped with a diverse mix of NVIDIA GPUs including H100, A100, V100, L4, and T4. Because hardware is deployed incrementally, serving systems must effectively utilize this heterogeneous mix rather than relying solely on a single generation of homogeneous accelerators.

Amazon Web Services. A similar trend is evident in AWS's infrastructure[4], which manages a highly heterogeneous fleet combining NVIDIA GPUs spanning multiple generations (e.g., H100, A100, and cost-effective A10G or L4 instances) with proprietary silicon such as Trainium and Inferentia.

These real-world deployments confirm that heterogeneous GPU environments are the norm rather than the exception in production LLM serving, validating the practical relevance of BOUTE's problem formulation.

[4] https://docs.aws.amazon.com/dlami/latest/devguide/gpu.html

F ADAPTING TO DISTRIBUTION SHIFTS

In production, the workload may evolve after deployment. We categorize distribution shifts by their required adaptation and associated overhead.

Table 6. Distribution shift scenarios and required adaptation.

Dynamic Scenario	Required Action	Overhead
Quality (request difficulty) shift	Online rescheduling	∼30s
Load (arrival rate) shift	Online rescheduling	∼30s
Resource availability change	Online rescheduling	∼30s
Request length distribution shift	Database extension	See below

Rationale. The offline database maps ⟨deployment, load⟩ to latency. Changes in quality distribution, load level, or resource availability affect only the routing threshold $\boldsymbol{\tau}$ and deployment choice $\mathbf{c}$, not the underlying latency characteristics stored in the database. Consequently, these shifts require only online rescheduling (∼30s, Table 2).

Request length distribution shift. When the input/output length distribution shifts significantly, the database must be extended with new ⟨deployment, load⟩ → latency entries under the updated distribution. We propose two complementary strategies to minimize overhead:

Strategy 1: Parallelized extension. BOUTE continues serving with the existing database while extension proceeds in the background. Since performance simulations of different deployment candidates are independent, the process is highly parallelizable. As shown in Table 7, parallelizing across 32 CPU threads reduces offline simulation time from 24.5 minutes to 54 seconds.

Table 7. Offline simulation time with multi-threaded parallelization.

CPU Thread(s)	1 (Baseline)	8	32
Simulation Time	24.5 min	3.4 min	54 s

Strategy 2: Pre-computed configuration space. Following established practices in LLM serving (e.g., Splitwise (Patel et al., 2024)), we can pre-profile a grid of anticipated input/output length distributions before deployment. When a length distribution shift occurs, the system performs a fast database lookup for the nearest pre-computed profile, avoiding any online simulation.

G CONFIGURATION SWITCHING

When BOUTE triggers a rescheduling event (e.g., due to a distribution shift), the system must transition from the current deployment to the updated configuration. We handle this through rolling updates: model replicas are incrementally reconfigured in batches while the remaining fleet continues serving requests. This approach, which follows

established industry practices for zero-downtime deployments, ensures service stability without transient latency spikes. The rescheduling decision itself (∼30s) is orders of magnitude faster than the rolling update process, so the optimization does not become a bottleneck during transitions.

H EXTENSIBILITY TO ADDITIONAL PARALLELISM SCHEMES

BOUTE's framework is designed to be extensible to parallelism strategies beyond TP and PP, with no fundamental limitations. The key enabler is the clean decoupling between parallelism enumeration (offline) and co-optimization search (online):

Offline phase: Extending the enumeration from (TP, PP) to include additional dimensions—such as expert parallelism (EP) for Mixture-of-Experts models or sequence parallelism (SP)—requires only adding the corresponding configurations to the simulator. The simulator processes these configurations and populates the performance database.

Online phase: The MOBO framework remains entirely unchanged, as it treats parallelism strategies as discrete deployment candidates. Additional parallelism dimensions simply expand the candidate set without requiring any algorithmic modifications.

In other words, new parallelism schemes are abstracted as additional entries in the performance database—the co-optimization logic remains entirely agnostic to the underlying parallelism type.

I HETEROGENEOUS DEPLOYMENT DESIGN RATIONALE

Combining homogeneous candidates via data parallelism across GPU types is an intentional design choice rather than an inherent limitation. Cross-GPU-type model parallelism (e.g., TP or PP spanning H100 and RTX 5090) would introduce severe performance bottlenecks, as heterogeneous GPUs in cloud environments typically lack high-bandwidth interconnects—for instance, inter-GPU bandwidth is often below 5 Gbps or entirely absent on platforms such as Google Cloud and RunPod, far below the requirements of model-parallel communication. Data parallelism across GPU types avoids this bottleneck entirely, as replicas operate independently with no inter-replica communication.

J INDIVIDUAL STRUCTURAL COMPONENT ABLATION

To isolate the contribution of each structural component, we conduct ablation experiments disabling S1 (load-fraction encoding), S4 (output transformation), and S5 (model-GPU preference encoding) individually.

Table 8. Scheduling time under individual structural component ablation.

GPU Types / Number	BOUTE	w/o S1	w/o S4	w/o S5	w/o all
4 / 32	32.5s	3.8 min	1.1 min	2.4 min	5.3 min
6 / 48	1.2 min	6.5 min	2.1 min	4.2 min	8.9 min

Each component contributes meaningfully: removing load-fraction encoding (S1) has the largest individual impact (∼7× slowdown), followed by model-GPU preference encoding (S5, ∼4×) and output transformation (S4, ∼2×). The combined removal produces an ∼8× slowdown, confirming that the performance gain stems from the integration of multiple problem-specific structural designs rather than any single technique.

K QUALITY SIGNAL AND DEPLOYMENT REALITY

BOUTE's optimization framework is agnostic to the specific quality metric employed. In practice, service providers commonly use application-dependent proxies such as LLM-as-a-Judge (as in our MT-Bench evaluation), task-specific accuracy, or user satisfaction scores. We acknowledge that proxy metrics may not fully correlate with user-perceived quality and can be noisy or biased. However, BOUTE remains practical under imperfect quality signals: it optimizes routing and deployment given whatever quality proxy is available, and when better measurements become available, the framework adapts accordingly by re-running the online optimization phase (∼30s). Improving quality measurement itself is an important but orthogonal research direction that applies to all quality-aware serving systems.

L BASELINE SELECTION RATIONALE

We select RouteLLM with uniform resource allocation to reflect current industry practice, where routing logic operates on pre-configured, static infrastructure. RouteLLM focuses solely on query-level routing and does not account for deployment-level allocation—precisely the gap that BOUTE addresses. The underutilization of small-model GPUs in high-quality regimes is therefore not a baseline weakness but a fundamental limitation of prior works that optimize routing in isolation. We enforce a static deployment constraint for RouteLLM to isolate BOUTE's core contribution: jointly optimizing routing and deployment.