SCOOT: SLO-Oriented Performance Tuning for LLM Inference Engines

Anonymous Author(s)

Abstract

As large language models (LLMs) are gaining increasing popularity across a wide range of web applications, it is of great importance to optimize service-level objectives (SLOs) for LLM inference services to enhance user satisfaction and improve the competitiveness of cloud vendors. In this paper, we observe that adjusting the parameters of LLM inference engines can improve service performance, and the optimal parameter configurations of different services are different. Therefore, we propose SCOOT, an automatic performance tuning system to optimize SLOs for each LLM inference service by tuning the parameters of the inference engine. SCOOT jointly exploits single-objective and multiple-objective Bayesian optimization (BO) techniques to handle various optimization objectives via exploration and exploitation. Moreover, SCOOT prunes the search space with known constraints and adopts a random forest to learn hidden constraints during the tuning process to mitigate invalid exploration. To improve the tuning efficiency, SCOOT utilizes the parallel suggestion to accelerate the tuning process. Extensive experiments demonstrate that SCOOT considerably outperforms existing tuning techniques in SLO optimization while greatly improving the tuning efficiency. SCOOT is universally applicable to various LLM inference engines and is easily expandable to new parameters. Currently, SCOOT has already been implemented in the production environment of a leading international technology company.

CCS Concepts

• **Computing methodologies** → **Parallel computing methodologies**; **Natural language generation**; **Machine learning**.

Keywords

Service-Level Objective, LLM Inference Engine, Performance Tuning, Bayesian Optimization, Cloud Computing

ACM Reference Format:

Anonymous Author(s). 2025. SCOOT: SLO-Oriented Performance Tuning for LLM Inference Engines. In *Proceedings of Proceedings of the ACM Web Conference 2025 (WWW '25).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

With the impressive capabilities demonstrated by large language models (LLMs) across various web applications [1], cloud vendors such as Alibaba Cloud and AWS have started offering services of LLM deployment and inference [2, 3]. Customers (e.g., developers and providers of web applications) can deploy specified LLMs on their exclusive computing resources. Through the application programming interface (API) provided by the cloud vendor, customers' requests can be continuously served by the deployed LLMs.

These LLM inference services run LLM instances with advanced inference engines such as vLLM [4] and TensorRT-LLM [5], which are equipped with cutting-edge technologies, such as continuous batching [6], paged attention [7], and chunked prefill [8]. These techniques can accelerate inference and improve throughput, thus delivering a high service performance to customers.

To ensure service performance, customers always agree on service level objectives (SLOs) with cloud vendors. SLOs are defined as a series of performance metric constraints, such as requiring that the 95th percentile latency of requests be less than 1 second. If cloud vendors violate these SLOs, they not only have to compensate customers but also face reputational damage. Therefore, appropriately setting SLOs is critical for cloud vendors. They should optimize SLOs (e.g., guarantee a lower tail latency) to improve customer satisfaction while ensuring that there won't be SLO violations.

Typically, cloud vendors stress test services with high request rates and set the performance metrics achieved under the stress testing as SLOs. This ensures that SLOs won't be violated even under heavy workload scenarios. By improving the service performance under stress testing, cloud vendors can deliver better SLOs to customers. In this paper, we observe that adjusting the parameters of LLM inference engines has great potential to improve service performance and the optimal parameter configurations of various LLM inference services are different. Therefore, we stand in the shoes of cloud vendors and optimize SLOs for each LLM inference service by tuning the parameters of LLM inference engines under stress testing. **The relevance of this paper to the systems and infrastructure for web is elaborated in Appendix A.1**.

Numerous performance tuning methods have been proposed and applied across various fields [9], but they all fall short of efficiency and optimality for tuning LLM inference engines. Methods like random sampling and meta-heuristic algorithms, such as Monte Carlo sampling [10] and genetic algorithms [11] lack efficiency as they fail to fully utilize historical information. Besides, heuristic searches rely on expert knowledge and elaborate pre-profiling to model relationships between parameters and performance. Nevertheless, with the rapid evolution of technologies in LLM inference engines, the parameters' numbers and ranges are frequently updated, which makes the modeled parameter-performance relationship outdated and the designed heuristic search methods inapplicable. Additionally, learning-based approaches such as reinforcement learning (RL) [12] and Bayesian optimization (BO) [13] have also been widely exploited in performance tuning. They can effectively leverage historical information and tune parameters automatically without

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WWW '25, April 28–2 May, 2025, Sydney, Australia

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

prior knowledge. However, RL requires a time-consuming training process to obtain a well-behaved tuning agent, while existing BO solutions fail to address the following three challenges.

Challenge 1: Various Optimization Objectives. Customers seek to enhance different performance metrics depending on their application requirements, such as improving request throughput for the periodically invoked offline recommendation application, reducing request tail latency for the online classification application, and minimizing time-to-first-token (TTFT) and time-per-output-token (TPOT) simultaneously for interactive applications such as chatbot, requiring the tuner to have the capability of handling both single-objective and multi-objective optimization problems.

Challenge 2: Complex Known and Hidden Constraints. Some parameters of inference engines depend on the settings of other parameters, thus causing constraints on the search space. For example, for vLLM, `max-num-batched-tokens` must be greater than or equal to `max-num-seqs`. We refer to these as known constraints, which can be provided to the tuner ahead of time. Besides, given a specific service, certain parameter combinations can lead to inference engine crashes during the stress testing. In this paper, these infeasible parameter combinations are referred to as hidden constraints. Different inference services have different hidden constraints that are initially unknown and must be learned throughout the tuning process. Specifically, for some certain services, vLLM often crushes due to a timeout error during the stress testing when `scheduler-delay-factor` is set to a large value.

Challenge 3: High Evaluation Overhead. Learning-based tuners always learn the correlation between the service performance and the engine's parameter settings by evaluating various parameter configurations. As stress testing an inference service takes about 5 to 10 minutes, even with only 30 evaluations for tuning, the total time spent tuning a single service can range from 2.5 to 5 hours. Since the increasing popularity of LLMs leads to the deployment of a large number of LLM inference services, the cumulative time required for tuning these services is prohibitive.

To tackle these challenges, we propose SCOOT, a **S**ervi**C**e-level **O**bjective **O**riented performance **T**uning system, which automatically tune parameters of LLM inference engines to optimize SLOs for LLM inference services. We first propose a general formulation of the inference engine tuning problem to accommodate various optimization objectives and complex constraints, and SCOOT can resolve the problem with BO, where single-objective BO (SOBO) and multi-objective BO (MOBO) [14–16] are respectively employed to search optimized parameter configurations for single-objective and multi-objective optimization scenarios, thus addressing challenge 1. Since constraint violations result in invalid observations caused by engine crashes, which greatly hurts the tuning efficiency, SCOOT prunes the search space with known constraints and exploits a random forest to learn hidden constraints during the tuning process, thus mitigating challenge 2. To resolve challenge 3, SCOOT employs the parallel suggestion technique to recommend multiple parameter configurations each time for simultaneous evaluation, thus fully utilizing idle computing resources to speed up tuning. SCOOT can support various inference engines and is easily expandable to new parameters. It's currently in use at Company-X[1].

[1]Company-X is the anonymous name of a technology company with billions of users.

Table 1: PARAMETERS TO TUNE.

Configuration Parameter	Type	Range
`tensor-parallel`	Integer	[1, #GPUs]
`max-num-seqs`	Integer	[64, 8192]
`max-num-batched-tokens`	Integer	[64, 8192]
`block-size`	Enumeration	{8, 16, 32}
`scheduler-delay-factor`	Float	[0, 2]
`enable-chunked-prefill`	Boolean	{`True`, `False`}
`enable-prefix-caching`	Boolean	{`True`, `False`}
`disable-custom-all-reduce`	Boolean	{`True`, `False`}
`use-v2-block-manager`	Boolean	{`True`, `False`}

We conduct extensive experiments with various LLMs and different types and numbers of GPUs under request traces collected from various LLM-based web applications at Company-X. The results show that SCOOT can speed up the tuning process and significantly optimize SLOs, improving the request throughput by up to 68.3%, reducing the request tail latency by up to 40.6%, and reducing the TTFT and TPOT by up to 99.8% and 61.0%, respectively, compared to the default parameter configuration and existing tuning methods. The main contribution of this paper is summarized as follows.

- To the best of our knowledge, this is the first study that introduces performance tuning into the field of LLM serving, and we uncover the significance of tuning LLM inference engines with real-world request traces.
- We propose a general formulation of the inference engine tuning problem that accommodates various optimization objectives and constraints, and we design SCOOT to solve the problem by intelligently searching optimized parameter configurations with BO.
- Random forest regression is employed by SCOOT to learn hidden constraints during the tuning process to avoid invalid explorations, while the parallel suggestion technique is adopted to significantly improve the tuning efficiency using additional computing resources.
- Extensive experiments are conducted to confirm the superiority of SCOOT in terms of both the optimality and efficiency for tuning LLM inference engines under various LLMs, computing resources, and request traces.

2 Background and Motivation

Background: parameters of the LLM inference engine. To provide flexibility of use, inference engines expose many parameters. For vLLM, these parameters include boolean variables such as `enable-chunked-prefill` that can enable the chunked prefill technique, integer and float variables such as `max-num-seqs` and `scheduler-delay-factor` that can change the request scheduling strategy, and enumeration variables such as `block-size` that can change the memory allocation policy. In this paper, we focus on tuning parameters that do not affect the accuracy of LLMs. Therefore, we do not consider parameters related to model compression, such as `quantization`. Besides, although speculative decoding has been theoretically proven not to hurt LLM accuracy [17], it still affects the LLMs' generation results, making it inapplicable for certain applications. Hence, we also do not tune parameters related to speculative decoding. In this paper, we choose vLLM as the inference engine, and the parameters to be tuned are listed in Table 1, which constructs a huge search space of billions of configuration points.

Table 2: VARIOUS SERVICE CHARACTERISTICS.

Service	Application	GPU Type	GPU Number	LLM
A	SQL	A100	2	LLAMA2-13B
B	**BOT**	A100	2	LLAMA2-13B
C	BOT	**A10**	2	LLAMA2-13B
D	BOT	A10	**4**	LLAMA2-13B
E	BOT	A10	4	**LLAMA2-7B**

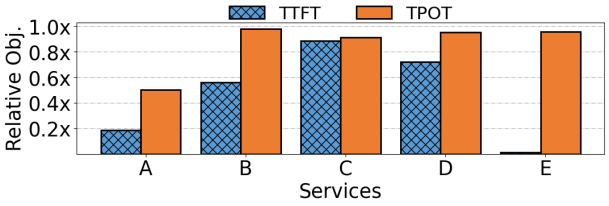


Figure 1: Optimal TTFT and TPOT for various services. The TTFT and TPOT shown are relative values compared to those of the default parameter configuration. The lower, the better.

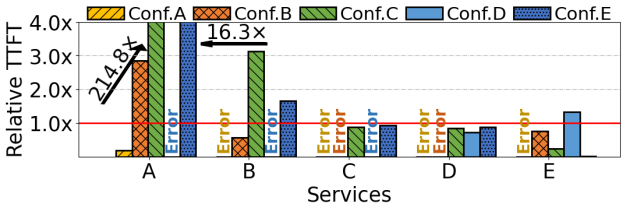
Motivation: parameter adjustment of the LLM inference engine can enhance performance for LLM inference services. We conduct experiments to find the optimal parameter configurations of vLLM with grid searches for five services of various LLMs deployed on different numbers and types of GPUs under request traces of various applications. The service characteristics in the experiments are shown in Table 2, where A10 and A100 separately represent the NVIDIA A10 24G GPU and NVIDIA A100 80G GPU, and SQL and BOT represent the request traces of text-to-SQL and chatbot applications at Company-X. Other experimental settings are the same as those in Section 5. Fig.1 presents the relative TTFT and TPOT performed by optimal parameter configurations for these services. We can observe that TTFT and TPOT can be reduced by up to 98.9% and 49.9% compared to the default configuration, respectively, which confirms the significance of performance tuning.

Motivation: different inference services' optimal parameter configurations are different. We conduct experiments to apply the optimal parameter configuration of a service to other services. Figure 2 presents the experimental results. We can observe that the performance exhibited by the optimal parameter configuration for a service may perform poorly or even violate hidden constraints and cause errors for other services. Hence, there is no one "best practice" configuration that works best in all scenarios, and it is essential to conduct performance tuning for each inference service.

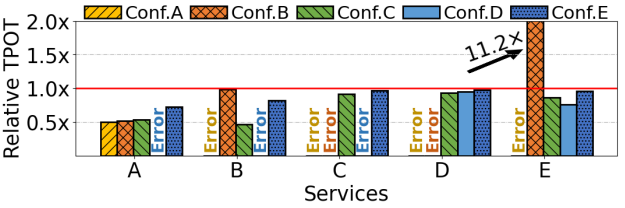
3 Problem Formulation

Given an inference service, the goal of performance tuning for the LLM inference engine is to find the optimal or near-optimal parameter configuration that maximizes the objective and satisfies known and hidden constraints. As we mentioned before, different customers want to optimize different performance metrics. Therefore, we provide a generalized problem formulation that supports various optimization objectives and complex constraints.

We define the configuration search space $\mathbf{\Lambda} = \Lambda_1 \times \Lambda_2 \times ... \times \Lambda_n$, where Λ_i represent the range of the ith parameter to tune. Vector $\boldsymbol{x} \in \mathbf{\Lambda}$ is utilized to denote a specific parameter configuration, where the ith element $\boldsymbol{x}[i] \in \Lambda_i$ corresponds to the value of the ith parameter. Moreover, we respectively use $T(\boldsymbol{x})$, $L(\boldsymbol{x})$, $\Phi(\boldsymbol{x})$, and $\Theta(\boldsymbol{x})$ to



(a) Relative time-to-first-token (TTFT) compared to the default configuration. The lower, the better.



(b) Relative time-per-output-token (TPOT) compared to the default configuration. The lower, the better.

Figure 2: TTFT and TPOT for applying optimal parameter configurations of different services to other services. Conf. X represents the optimal parameter configuration for the service $X \in \{A, B, C, D, E\}$. The red lines (1.0×) in Sub-figures (a) and (b) indicate the TTFT and TPOT under the default parameter configuration of the inference engine, respectively.

represent the functions of request throughput, tail latency, average TTFT, and average TPOT for the inference service under configuration $\boldsymbol{x}$. We leverage $C = \{c_1, c_2, ..., c_m\}$ to symbolize the set of known constraints, where c_i is the ith known constraint which restricts the relationship between elements of $\boldsymbol{x}$. For example, a known constraint can be expressed as "$\boldsymbol{x}[1] < \boldsymbol{x}[3]$ if $\boldsymbol{x}[4]$ is False". Moreover, we utilize $POF(\boldsymbol{x})$ to represent $\boldsymbol{x}$'s probability of feasibility (POF) that $\boldsymbol{x}$ won't violate hidden constraints. Therefore, the tuning problem can be formulated as

$$\mathbb{P}: \max_{\boldsymbol{x} \in \mathbf{\Lambda}} \quad \lambda_t \cdot T(\boldsymbol{x}), \lambda_l \cdot L(\boldsymbol{x}), \lambda_\phi \cdot \Phi(\boldsymbol{x}), \lambda_\theta \cdot \Theta(\boldsymbol{x}) \tag{1}$$

s. t.

$$c_i, \forall c_i \in C, \tag{2}$$

$$POF(\boldsymbol{x}) \geq \Delta, \tag{3}$$

where Eq. (2) and Eq. (3) respectively denotes the known and hidden constraints. Δ is the POF threshold used to avoid violations of hidden constraints. $\lambda_t \in \{0, 1\}$ and $\lambda_l, \lambda_\phi, \lambda\theta \in \{0, -1\}$ are utilized to control the optimization objectives. If $(\lambda_t, \lambda_l, \lambda_\phi, \lambda_\theta) = (1, 0, 0, 0)$, the optimization objective is to only maximize the request throughput, which is applicable to periodically invoked applications such as offline recommendation. If $(\lambda_t, \lambda_l, \lambda_\phi, \lambda_\theta) = (0, -1, 0, 0)$, the optimization objective is to only minimize the tail latency, which is appropriate for non-interactive online applications such as classification. If $(\lambda_t, \lambda_l, \lambda_\phi, \lambda_\theta) = (0, 0, -1, -1)$, the optimization objective is to simultaneously minimize the TTFT and TPOT, which is suitable for interactive applications.

4 SCOOT: Solution Description

We present SCOOT's design, including its use of SOBO and MOBO for solving $\mathbb{P}$, its innovative features of parallel suggestion and random forest-based POF learning, and the SLO robustness assurance.

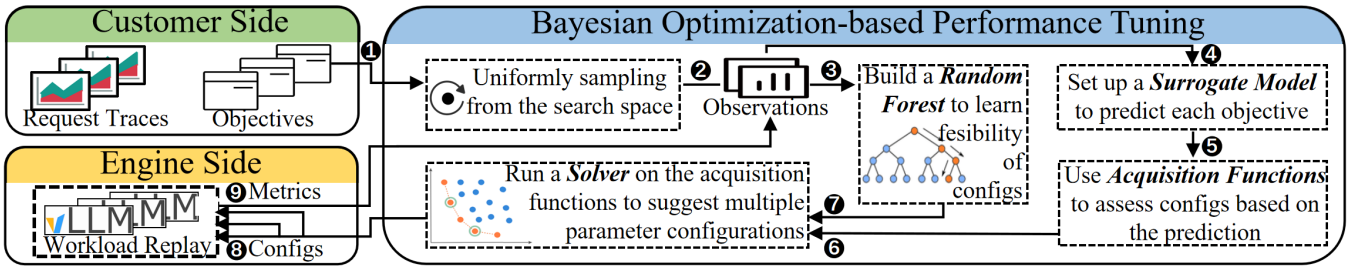


Figure 3: SCOOT workflow. SCOOT leverages BO to find optimized parameter configurations via exploration and exploitation.

4.1 SCOOT Workflow

The workflow of SCOOT is depicted in Fig. 3, which consists of nine key steps to find optimized parameter configurations. Customers define their optimization objectives, and SCOOT ❶ gathers their request traces that include both input text of requests and output text generated by the LLM. Then, SCOOT leverages Sobol sequence-based Quasi-Monte Carlo [18] to uniformly ❷ sample configurations across the search space, where the number of samples matches the search space's dimensionality. Then, SCOOT runs the inference engine with each sampled configuration and stress tests the inference service to obtain initial observations. These observations of configuration-performance pairs are leveraged to ❸ build a random forest and ❹ construct a surrogate model to learn the $POF(\cdot)$ and predict the probability distribution of each optimization objective, respectively. Subsequently, acquisition functions are exploited to ❺ assess configurations according to the predicted results. Based on the assessment, a solver ❻ suggests multiple configurations in parallel while adhering to the known constraints and ❼ ensuring hidden constraints using $POF(\cdot)$ learned by the random forest. Lastly, the inference engine ❽ is started with each suggested parameter configuration, and the stress testing is conducted to ❾ obtain new observations to refine the random forest and the surrogate model. Steps ❸~❾ run iteratively, and the tuning process stops until the number of observations reaches a given threshold.

In the workflow, the LLM, GPU number, and GPU type used for tuning are the same as the inference service owned by the customer.

4.2 Bayesian Optimization-based Solution

BO is a theoretically grounded method for finding the optimum of black-box functions. It can explore the complex multi-dimensional search space efficiently and intelligently [13], which is suitable for solving problems with expensive evaluation overhead. BO leverages a surrogate model to approximate the objective function and use acquisition functions to assess configuration points for suggesting.

SCOOT leverages SOBO to maximize throughput and minimize tail latency for non-interactive offline and online applications, respectively. For interactive applications, it exploits MOBO to minimize TTFT and TPOT simultaneously by finding a set of parameter configurations representing the Pareto frontier that denotes the optimal trade-offs between TTFT and TPOT. We do not linearly combine TTFT and TPOT as a single optimization objective and solve it with SOBO because TTFT can be hundreds or even thousands of times larger than TPOT. Hence, it is difficult to assign weights to TTFT and TPOT to make a good trade-off. Besides, TTFT and TPOT are always two conflicting optimization objectives as illustrated in Appendix A.2.

4.3 Surrogate Model

The surrogate model of BO predicts the objective function $f(\boldsymbol{x})$ based on observations. It models $f(\boldsymbol{x})$ for a given configuration $\boldsymbol{x}$ as a random variable and predicts its probability distribution. In the context of LLM inference engine tuning, $f(\boldsymbol{x})$ can represent the objective functions of request throughput $T(\boldsymbol{x})$, request tail latency $L(\boldsymbol{x})$, average TTFT $\Phi(\boldsymbol{x})$, and average TPOT $\Theta(\boldsymbol{x})$.

SCOOT uses the Gaussian process (GP) as the surrogate model. Given a parameter configuration $\boldsymbol{x}$, GP assumes that the probability distribution of $f(\boldsymbol{x})$ follows a Gaussian distribution whose mean $\mu(f(\boldsymbol{x}))$ and the variance $\sigma^2(f(\boldsymbol{x}))$ are respectively computed by

$$\mu(f(\boldsymbol{x})) = k(\boldsymbol{x}, \boldsymbol{X})(k(\boldsymbol{X}, \boldsymbol{X}) + \tau^2 I)^{-1}\boldsymbol{y}, \tag{4}$$

$$\sigma^2(f(\boldsymbol{x})) = k(\boldsymbol{x}, \boldsymbol{x}) - k(\boldsymbol{x}, \boldsymbol{X})(k(\boldsymbol{X}, \boldsymbol{X}) + \tau^2 I)^{-1}k(\boldsymbol{X}, \boldsymbol{x}), \tag{5}$$

where $\boldsymbol{X}$ represents the previously evaluated configurations, $\boldsymbol{y}$ denotes the corresponding observed objective values, τ^2 is the level of white noise, and $k(\boldsymbol{x}, \boldsymbol{x}')$ is the covariance function that quantifies the similarity between input points $\boldsymbol{x}$ and $\boldsymbol{x}'$ for inferring the relationships between their objective function values. SCOOT employs the Matern kernel ($\frac{3}{2}$) with input wrapping [19] as the covariance function due to its capability to balance the smoothness and flexibility when modeling unknown functions. SCOOT utilizes maximum likelihood estimation [20] to learn τ^2 during tuning.

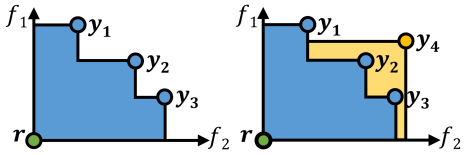
For SOBO, a single-output GP is leveraged to predict the objective function. For MOBO, SCOOT adopts a multi-output GP by considering each output to be independent. During the tuning process, the prediction accuracy of the GP model is continuously improved as more observations are collected.

4.4 Acquisition Function

The acquisition function of BO assesses parameter configurations in the search space, which calculates a score for each configuration point $\boldsymbol{x}$ according to the surrogate model's predicted mean $\mu(f(\boldsymbol{x}))$ (indicating the expected performance) and variance $\sigma^2(f(\boldsymbol{x}))$ (representing uncertainty). Parameter configurations with high scores are more likely to be suggested for evaluation. Different acquisition functions balance exploration (visiting areas with high uncertainty) and exploitation (intensively searching areas with good known objective values) in different manners.

4.4.1 ***SOBO Acquisition Function.*** For SOBO, commonly used acquisition functions include upper confidence bound (UCB), probability of improvement (PI), and expected improvement (EI). UCB incorporates both mean and variance into the score and prioritizes the point with a high balance of exploration and exploitation through a trade-off parameter β, which is expressed by

$$UCB(\boldsymbol{x}) = \mu(f(\boldsymbol{x})) + \beta \cdot \sigma(f(\boldsymbol{x})), \tag{6}$$



(a) Hypervolume (HV). (b) HV improvement.

Figure 4: Illustration of two-dimensional HV. f_1 and f_2 are two objective functions and r is the reference point. (a) Blue area represents the HV of the existing solution set. (b) Yellow area depicts the HV improvement after adding y_4.

where $\sigma(f(\boldsymbol{x})) = \sqrt{\sigma^2(f(\boldsymbol{x}))}$ represents the standard deviation.

PI prioritizes the point that is likely to yield an improvement over the current known best observation, which is expressed by

$$PI(\boldsymbol{x}) = P(f(\boldsymbol{x}) + \xi > f(\boldsymbol{x}^+)), \tag{7}$$

where $\boldsymbol{x}^+$ is the point that has the largest objective function value known so far, and ξ is leveraged to encourage exploration.

EI not only considers the probability of objective improvement but also the magnitude of the expected improvement, which is computed by

$$EI(\boldsymbol{x}) = \mathbb{E}(\max(0, f(\boldsymbol{x}) + \xi - f(\boldsymbol{x}^+))), \tag{8}$$

where $\boldsymbol{x}^+$ represents the best observed configuration point as well, and ξ is also a parameter used to encourage exploration.

4.4.2 ***MOBO Acquisition Function.*** For MOBO, expected hypervolume improvement (EHVI) is widely adopted as the acquisition function to identify promising parameter configurations that denote optimal trade-offs between multiple optimization objectives. EHVI assesses a new configuration point based on the potential improvement it might bring to the hypervolume of the existing solution set, which is calculated by

$$EHVI(\boldsymbol{x}) = \mathbb{E}(\max(0, HV(\boldsymbol{\mathcal{Y}} \cup \{f(\boldsymbol{x})\}) - HV(\boldsymbol{\mathcal{Y}}))), \tag{9}$$

where $HV(\cdot)$ is the hypervolume function and $\boldsymbol{\mathcal{Y}}$ denotes the objective values of previously evaluated configurations.

The hypervolume function $HV(\cdot)$ is calculated by measuring the volume of space enclosed between the Pareto frontier and a reference point $\boldsymbol{r}$, where $\boldsymbol{r}$ is typically a lower bound point for all objective functions. When there are only two optimization objectives, $HV(\cdot)$ is calculated by summing the areas of rectangles formed between each point on the Pareto frontier and $\boldsymbol{r}$. Figure 4 presents how the two-dimensional $HV(\cdot)$ is computed.

A larger $HV(\cdot)$ value indicates a Pareto front that covers a larger objective space, offering a broader range of promising solutions. By maximizing the EVHI, the search process can be guided towards configurations that can improve the overall quality and diversity of the Pareto frontier, thus optimizing multiple objectives simultaneously and providing more optional configurations for customers.

4.5 Configuration Suggestion

4.5.1 ***SOBO Suggestion.*** For SOBO, a common practice is to select a specific acquisition function and suggest parameter configurations that maximize the acquisition function. However, when it comes to real-world applications, the selected acquisition function might work sub-optimally for the tuning task, and the best acquisition function is challenging to identify beforehand [21].

To mitigate this issue, SCOOT employs multi-objective acquisition function ensemble (MACE) [22]. MACE uses three acquisition functions, UCB, PI, and EI, to assess configuration points and runs a solver to find out the Pareto frontier that denotes the optimal trade-offs of the three acquisition functions. Suggested configurations are randomly selected in the Pareto frontier. With MACE, SCOOT can avoid using sub-optimal acquisition functions all the time, thereby improving the quality of suggested parameter configurations.

During the tuning process, for UCB, β is dynamically adjusted, and in the tth tuning iteration, β is calculated by $2log(\frac{t^2\pi^2}{6\delta})$, where $\delta = \frac{1}{t^2}$ [23]. For EI and PI, ξ is set to a fixed value of 0.0001.

4.5.2 ***MOBO Suggestion.*** For MOBO, SCOOT leverages the EHVI as the acquisition function and runs a solver to suggest parameter configurations by maximizing EHVI. The default configuration of the inference engine is chosen as the reference point $\boldsymbol{r}$.

4.5.3 ***Known Constraint Assurance.*** For both SOBO and MOBO, in the process of solvers solving optimization problems, the solution space is pruned by known constraints.

4.6 Parallel Configuration Suggestion

With the popularity of LLMs, an increasing number of LLM inference services are being deployed, and tuning them incurs substantial time overhead. To mitigate this problem, SCOOT suggests multiple parameter configurations at a time and utilizes multiple sets of computing resources to conduct stress testing in parallel.

Given the parallelism degree k, for SOBO, SCOOT randomly selects k points from the Pareto frontier of the three acquisition functions for suggesting. For MOBO, the top-k points with the largest EHVI are suggested. Thus, tuning can be accelerated by a factor of k when the total observation number is fixed.

Although the parallel suggestion requires additional computing resources to evaluate configurations in parallel, many computing resources in production clusters are often idle during off-peak hours such as nights and weekends, which can be conveniently used for tuning LLM inference engines.

4.7 Random Forest-based POF Learning

To handle hidden constraints, a common practice is to assign penalty objective values to the infeasible parameter configurations that cause engine crashes. However, it is challenging to set appropriate penalty objective values. Besides, penalty objective values often hurt the surrogate model [24]. Therefore, SCOOT leverages random forest regression to learn the $POF(\cdot)$ and handle hidden constraints by restricting the solver to adhere to Eq. (3) in the process of configuration suggestion. In this way, SCOOT can substantially reduce invalid observations without affecting the surrogate model.

For the hidden constraint expressed in Eq. (3), the feasibility probability threshold Δ is critical. If Δ is fixed, setting it too high may result in repeatedly suggesting configuration points near already observed feasible configurations, potentially missing out on other superior configuration points. Conversely, setting Δ too low may lead to suggestions of infeasible configurations. Hence, SCOOT dynamically adjusts the threshold Δ during the tuning process.

The dynamic adjustment process of Δ can be found in Algorithm 1, where we can observe that when infeasible parameter configurations are suggested, Δ is increased to reduce the likelihood of

Algorithm 1: SCOOT: BO-based Performance Tuning

Input: N: total observation number; k: parallelism degree;

```
// Initialize observations
O ← UNIFORM_SAMPLE_AND_EVALUATE()
RF ← TRAIN(O) // Train random forest regression
Δ ← 0.5; c ← 0; v ← 0.05
while |O| < N do
    // Suggest k configurations
    {x_1, ..., x_k} ← SUGGEST(k, O, RF, Δ)
    // Collect performance metrics
    {y_1, ..., y_k} ← EVALUATE({x_1, ..., x_k})
    // Adjust feasibility threshold
    if there are invalid y in {y_1, ..., y_k} then
        Δ ← min(0.75, max(0.5, Δ + v))
        c ← 0 // Clear feasible suggestion count
    else
        c ← c + k
        if c ≥ 5 then
            Δ ← max(0.25, Δ − v)
            c ← c − 5
    // Update observations
    O ← UNION(O, {x_1, ..., x_k}, {y_1, ..., y_k})
    RF ← TRAIN(O) // Refine Random Forest
```

further suggesting infeasible configurations. Besides, after continuously suggesting feasible configurations five times, Δ is decreased to allow the discovery of potential superior configurations. Additionally, during the tuning process, the random forest is continuously refined using the latest observations to enhance prediction accuracy, ensuring that hidden constraints are intelligently followed without compromising the quality of performance tuning.

Since known and hidden constraints pure the solution space, solvers may be unable to give a sufficient number of solutions during the configuration suggestion. In such scenarios, SCOOT randomly samples configuration points with Sobol sequence-based Quasi-Monte Carlo to make up for the shortfall.

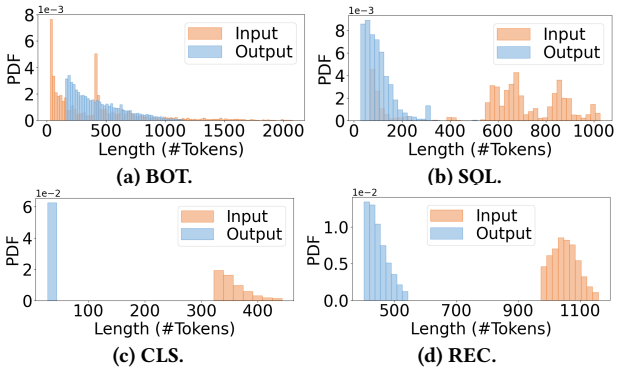
4.8 Ensuring SLO Robustness

Through extensive experiments, we find that when the performance tuning is conducted under different request arrival orders, the best parameter configurations are almost the same, but the corresponding optimized SLOs vary. Thus, after performance tuning, we conduct stress testing multiple times again using the best configuration, varying request arrival orders, and select the worst-case objectives as the SLOs to ensure SLO Robustness. Finally, we summarize the overall tuning procedure of SCOOT in Algorithm 1.

5 Experimental Evaluation

5.1 Experiment Setup

We implement SCOOT on top of HEBO [21], where the random forest regression is implemented using the sklearn [25] library. Request traces are collected from four LLM inference services at Company-X, including applications of text-to-SQL (**SQL**), chatbot (**BOT**), classification (**CLS**), and recommendation (**REC**). Since the computational load of an LLM inference request depends on its input and output lengths, to enhance the robustness of optimized SLOs, SCOOT uses the most loaded 50% of requests for stress testing, where the input and output lengths are presented in Fig. 5. For SQL and BOT, TTFT and TPOT are optimization objectives at the same time. For CLS and REC, 95th percentile latency and request throughput are utilized as optimization objectives, respectively.



(a) BOT. (b) SQL. (c) CLS. (d) REC.

Figure 5: Probability density function (PDF) of the request input and output lengths for four application request traces.

Multiple sets of computing resources are utilized in experiments, including **2A10**, **4A10**, **2A100**, and **4A100**, where A10 represents the NVIDIA A10 24GB GPU, A100 denotes the NVIDIA A100 80GB GPU, and the integer before the GPU type indicate the number of GPUs. For each set of computing resources, 256GB CPU memory and one 2.90GHz 32-core Intel(R) Xeon(R) CPU are equipped. Besides, A100 GPUs are connected over NVLink, and A10 GPUs are connected over PCIE. We use LLAMA2-7B and LLAMA2-13B [26] in experiments because the model architecture of LLAMA is representative and fine-tuned 7B and 13B LLMs can fulfill the requirements for most applications in practice. We adopt vLLM as the inference engine to tune due to its high usability and popularity.

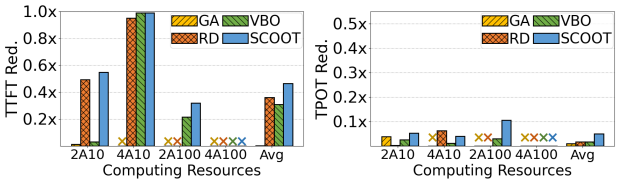
In each configuration evaluation, when the inference engine is started with the suggested configuration, we send requests from the request trace for 100 seconds and the request arrival times are generated using Poisson distribution with various request rates. For A10, the request rates for SQL, BOT, CLS, and REC are 5,5, 10, and 15, respectively. For A100, the request rates of these request traces are twice that of A10. The request rate is set according to the actual workload of the applications as well as considering the computing capability and GPU memory capacity of GPUs.

We compare SCOOT with three baselines, including random sampling (**RD**), genetic algorithm (**GA**), and Vanilla BO (**VBO**). Detailed baseline description is provided in Appendix A.3.

5.2 SLO Optimization

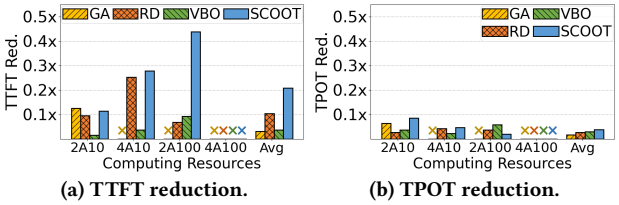
In the experiments of SLO optimization, we limit the total observation number to 30 and set the suggestion parallelism degree (PD) of SCOOT to one. The experimental results are presented in Figures 6, 7, 8, 9, 10, and 11, where we not only present the SLO improvement under each set of computing resources but also the average SLO improvement across various computing resources. The bar represents the SLO improvement compared to the default parameter configuration, and **the higher the bars, the better**. In addition, × represents that no better configuration than the default configuration is found.

For BOT and SQL applications, TTFT and TPOT are optimized simultaneously, and a Pareto frontier is resolved. We choose SLOs



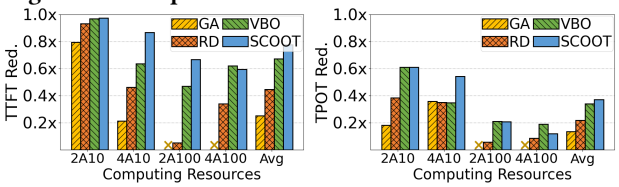
(a) TTFT reduction. **(b) TPOT reduction.**

Figure 6: SLO optimization for BOT under LLAMA2-7B.



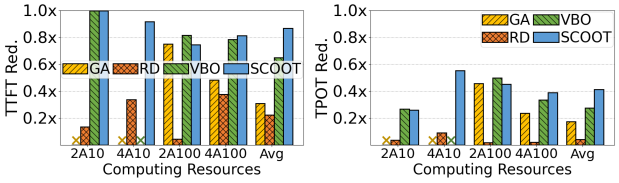
(a) TTFT reduction. **(b) TPOT reduction.**

Figure 7: SLO optimization for BOT under LLAMA2-13B.



(a) TTFT reduction. **(b) TPOT reduction.**

Figure 8: SLO optimization for SQL under LLAMA2-7B.



(a) TTFT reduction. **(b) TPOT reduction.**

Figure 9: SLO optimization for SQL under LLAMA2-13B.

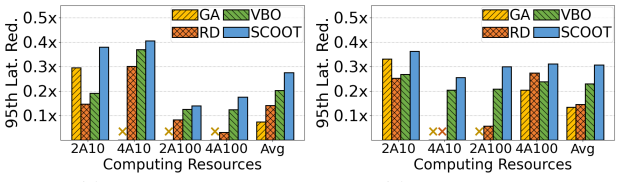
of the configuration with maximal HV from the Pareto frontier as the tuning result to report since it represents the best trade-off between TTFT and TPOT. HV of a single configuration point $\boldsymbol{x}$ is calculated by

$$HV(\boldsymbol{x}) = \max(0, y_{\boldsymbol{x}}^{\phi} - y_{\boldsymbol{r}}^{\phi}) \cdot \max(0, y_{\boldsymbol{x}}^{\theta} - y_{\boldsymbol{r}}^{\theta}), \tag{10}$$

where $y_{\boldsymbol{x}}^{\phi}$, $y_{\boldsymbol{x}}^{\theta}$, $y_{\boldsymbol{r}}^{\phi}$, and $y_{\boldsymbol{r}}^{\theta}$ denote the observed TTFT and TPOT of $\boldsymbol{x}$ and the reference point $\boldsymbol{r}$, respectively.

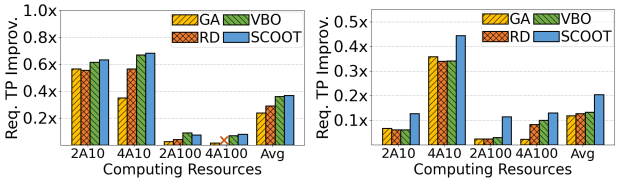
Figures 6, 7, 8, and 9 show that compared to the default configuration and baselines, SCOOT decreases the TTFT and TPOT by up to 98.9% and 10.5% for the BOT application, respectively, and reduces the TTFT and TPOT by up to 99.8% and 61.0% for the SQL application, separately. In addition, SCOOT surpasses all the baselines in terms of average TTFT and TPOT reduction across various computing resources for BOT and SQL applications, which confirms SCOOT's superiority in multi-objective optimization.

Figures 10 and 11 present that compared to the default configuration and baselines, SCOOT reduces request tail latency by up to 40.6% for the CLS application and increases request throughput by up to 68.3% for the REC application. Besides, SCOOT outperforms all the baselines in terms of average tail latency reduction and request throughput improvement across various computing resources for CLS and REC applications, respectively, which confirms SCOOT's superiority in single-objective optimization.



(a) LLAMA2-7B. **(b) LLAMA2-13B.**

Figure 10: Tail latency reduction for CLS.



(a) LLAMA2-7B. **(b) LLAMA2-13B.**

Figure 11: Request TP improvement for REC.

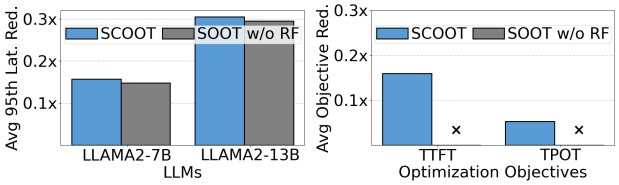
While in rare situations, SCOOT failed to outperform certain baselines in SLO optimization, SCOOT's tuning performance is more consistent. As shown in Fig. 9, in the case of 2A100, VBO slightly outperforms SCOOT in both TTFT and TPOT reduction. However, in the case of 4A10, VBO fails to find a configuration that is better than the default configuration, while SCOOT greatly reduces TTFT and TPOT. Such a consistently superior performance makes SCOOT outperform all the baselines in the average performance of SLO optimization for various LLMs and applications.

GA optimizes around the initially sampled configurations. With a limited number of observations, both the population size and the number of iterations of GA are restricted to a small value. Thus, GA not only lacks diversity in solutions but also cannot fully utilize historical information to guide the search process, and GA can hardly tackle constraints, hence often performing poorly.

RD is unable to utilize historical information to guide the sampling process and can not handle constraints. However, due to the uniform sampling, RD can explore a large range in the search space and hence has a higher diversity in sampled configurations than GA. Hence, RD often performs better than GA.

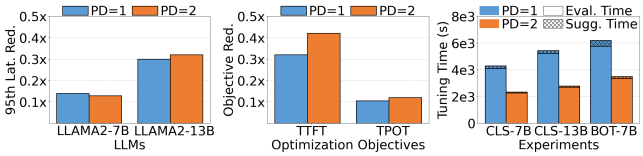
By leveraging BO to intelligently utilize historical observations to guide the tuning process, VBO often outperforms RD and GA in SLO optimization. However, by assigning a penalty objective value to unfeasible configurations, the surrogate model will confuse points near infeasible configurations with points that are feasible but perform poorly. This can lead to a lack of subsequent exploration in the adjacent area of infeasible configurations, thus missing out on some potentially superior solutions.

SCOOT's outstanding performance primarily comes from the employment of BO, which efficiently resolves black-box optimization problems, and the random forest-based POF learning, which



(a) Tail latency reduction for CLS under LLAMA2-7B & 13B. **(b) TTFT & TPOT reduction for BOT under LLAMA2-7B.**

Figure 12: Effectiveness of random forest-based POF learning.



(a) Tail latency reduction for CLS under LLAMA2-7B & 13B. **(b) TTFT & TPOT reduction for BOT under LLAMA2-7B.** **(c) Performance tuning time for CLS and BOT.**

Figure 13: Tuning efficiency of the parallel suggestion.

intelligently handles hidden constraints without affecting the GP-based surrogate model. Besides, by dynamically adjusting the POF threshold Δ, SCOOT can explore the region near the infeasible configurations after continuous exploration of feasible configurations so as to fully explore the solution space.

5.3 Ablation Studies

We conduct experiments to confirm the effectiveness of the random forest (RF)-based learning of hidden constraints under 2A100 and 4A100. The average SLO optimization results across various computing resources are presented in Fig. 12. Since requests from the CLS application seldom cause engine crashes, not applying RF-based POF learning has little impact on SCOOT's performance. However, since BOT application requests cause many hidden constraints, RF-based POF learning can greatly enhance SLO optimization.

5.4 Tuning Efficiency

We conduct experiments to tune the CSL and BOT applications under 2A100 with different parallelism degrees (PDs). In the experiments, the number of observations is also limited to 30 when PD is set to 2. Figures 13a and 13b show the SCOOT's performance in SLO optimization under various PDs, while Fig. 13c presents the total tuning time under various PDs.

Figures 13a and 13b show that when PD is set to 2, using the parallel suggestion achieves the same performance in SLO optimization as not using it. Besides, as presented in Fig. 13c, when PD is set to 2, the parallel suggestion technique significantly reduces the total tuning time, nearly cutting it in half, which demonstrates that the parallel suggestion can effectively accelerate the tuning process by PD times. Furthermore, Figure 13c shows that compared with the time of configuration evaluation (i.e., stress testing), the time required for configuration suggestion is negligible, which validates the efficiency of applying BO to tune LLM inference engines.

We also increase PD and conduct experiments. However, we find that when PD is set to a large value, such as greater than 4, SCOOT's performance of SLO optimization is compromised, and increasing the total observation number can mitigate this issue. In the future, we will explore the best number of observations under different PDs to find the most cost-effective acceleration solution.

5.5 Solution Quality

For multi-objective optimization, we evaluate the quality of solutions from two perspectives. First, the optimality of solutions, that is, whether the optimized parameter configurations lie on the Pareto frontier of TTFT and TPOT. Second, we focus on the diversity of solutions, that is, whether the range of optimized parameter configurations on the Pareto frontier is sufficiently broad to offer customers promising optional configurations to select based on their TTFT and TPOT requirements. We illustrate the resolved Pareto frontier of SCOOT and other baselines in Appendix A.4, which confirms the superiority of SCOOT in both the optimality and diversity of SLO optimization in multi-objective situations.

6 Related Works

6.1 Advanced LLM Inference Techniques

Existing inference engines such as vLLM and TensorRT-LLM support a variety of advanced inference techniques to improve the speed and throughput of LLM inference, including a series of kernel-level optimizations such as continuous batching [6], paged attention [7], flash attention [27, 28], and chunked prefill [8]. Besides, prefix caching [29] is employed by inference engines to reuse computed key and value tensors of requests' common prefix texts to reduce redundant memory allocation and computation for newly arrived requests, which can significantly improve the serving efficiency when requests share a long common prefix. Moreover, speculative decoding [17, 30–33] is also adopted by inference engines to accelerate LLM inference, where an efficient draft model is leveraged to generate tokens, and the LLM occasionally refines the draft.

Some of these technologies are effective in specific scenarios and have been implemented as configurable parameters of inference engines. However, how to combine these technologies to fully exploit the capabilities of inference engines has not been adequately studied, a gap that this study aims to fill.

6.2 Automatic Performance Tuning Approaches

Performance tuning techniques have been widely applied across a wide variety of fields, including database tuning [34–36], Spark configuration tuning [37–40], compiler optimization [24], and tuning of web-relevant applications [41, 42]. The vast majority of these performance tuning studies adopt BO to find optimized parameter configurations since BO is theoretically grounded and can efficiently learn the relationship between performance and parameters from evaluations, thus intelligently tuning parameters for performance improvement. Because of these advantages, BO is also employed in resource allocation studies [43–45].

To the best of our knowledge, previous studies have not focused on tuning LLM inference engines and can not address all the three unique challenges faced in inference engine tuning.

7 Conclusion and Future Works

In this paper, we propose SCOOT, a BO-based automatic inference engine tuning system, to optimize SLOs for LLM inference services. SCOOT leverages both SOBO and MOBO to handle various optimization objectives and utilizes random forest regression to learn hidden constraints during the tuning process. Additionally, SCOOT exploits the parallel suggestion to accelerate tuning with additional computing resources. Experimental results show that SCOOT can effectively speed up tuning and significantly optimize SLOs, improving the request throughput by up to 68.3%, reducing the request tail latency by up to 40.6%, and reducing the TTFT and TPOT by up to 99.8% and 61.0%, respectively. In the future, we will attempt to utilize meta-learning approaches to further accelerate the tuning process for new LLM inference services using the knowledge previously learned from tuning other inference services.

References

[1] Bonan Min, Hayley Ross, Elior Sulem, et al. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Computing Surveys*, 56(2):1–40, 2023.

[2] Alibaba cloud bailian platform. https://www.aliyun.com/product/bailian, 2024.

[3] Aws sagemaker. https://aws.amazon.com/sagemaker/, 2024.

[4] vllm. https://github.com/vllm-project/vllm, 2024.

[5] Tensorrt-llm. https://github.com/NVIDIA/TensorRT-LLM, 2024.

[6] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, et al. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.

[7] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, et al. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[8] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, et al. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134, 2024.

[9] Herodotos Herodotou, Yuxing Chen, and Jiaheng Lu. A survey on automatic parameter tuning for big data processing systems. *ACM Computing Surveys (CSUR)*, 53(2):1–37, 2020.

[10] Tito Homem-de Mello and Güzin Bayraksan. Monte carlo sampling-based methods for stochastic optimization. *Surveys in Operations Research and Management Science*, 19(1):56–85, 2014.

[11] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia tools and applications*, 80:8091–8126, 2021.

[12] Sindhu Padakandla. A survey of reinforcement learning algorithms for dynamically varying environments. *ACM Computing Surveys (CSUR)*, 54(6):1–25, 2021.

[13] Xilu Wang, Yaochu Jin, Sebastian Schmitt, and Markus Olhofer. Recent advances in bayesian optimization. *ACM Computing Surveys*, 55(13s):1–36, 2023.

[14] Nazan Khan, David E Goldberg, and Martin Pelikan. Multi-objective bayesian optimization algorithm. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 684–684, 2002.

[15] Kaifeng Yang, Michael Emmerich, André Deutz, and Thomas Bäck. Multi-objective bayesian global optimization using expected hypervolume improvement gradient. *Swarm and evolutionary computation*, 44:945–956, 2019.

[16] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization. *Advances in Neural Information Processing Systems*, 33:9851–9864, 2020.

[17] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[18] Russel E Caflisch. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 7:1–49, 1998.

[19] Jasper Snoek, Kevin Swersky, Rich Zemel, and Ryan Adams. Input warping for bayesian optimization of non-stationary functions. In *International conference on machine learning*, pages 1674–1682. PMLR, 2014.

[20] Matthias Seeger. Gaussian processes for machine learning. *International journal of neural systems*, 14(02):69–106, 2004.

[21] Alexander I Cowen-Rivers, Wenlong Lyu, Rasul Tutunov, et al. Hebo: Pushing the limits of sample-efficient hyper-parameter optimisation. *Journal of Artificial Intelligence Research*, 74:1269–1349, 2022.

[22] Shuhan Zhang, Fan Yang, Changhao Yan, et al. An efficient batch-constrained bayesian optimization approach for analog circuit synthesis via multiobjective acquisition ensemble. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(1):1–14, 2021.

[23] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning*, pages 1015–1022, 2010.

[24] Erik Orm Hellsten, Artur Souza, Johannes Lenfers, et al. Baco: A fast and portable bayesian compiler optimization framework. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, pages 19–42, 2023.

[25] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[26] Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models. 2023.

[27] Tri Dao, Dan Fu, Stefano Ermon, et al. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[28] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2023.

[29] Nikhil Jha and Kevin Wang. Improving large language model throughput with efficient long-term memory management.

[30] Heming Xia, Tao Ge, Peiyi Wang, et al. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3909–3925, 2023.

[31] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949, 2024.

[32] Sehoon Kim, Karttikeya Mangalam, Suhong Moon, et al. Speculative decoding with big little decoder. *Advances in Neural Information Processing Systems*, 36, 2024.

[33] Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, et al. Spectr: Fast speculative decoding via optimal transport. *Advances in Neural Information Processing Systems*, 36, 2024.

[34] Xinyi Zhang, Hong Wu, Zhuo Chang, et al. Restune: Resource oriented tuning boosted by meta-learning for cloud databases. In *Proceedings of the 2021 international conference on management of data*, pages 2102–2114, 2021.

[35] Jiale Lao, Yibo Wang, Yufei Li, et al. Gptuner: A manual-reading database tuning system via gpt-guided bayesian optimization. 2023.

[36] Tiannuo Yang, Wen Hu, Wangqi Peng, et al. Vdtuner: Automated performance tuning for vector data management systems. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 4357–4369, 2024.

[37] Anastasios Gounaris, Georgia Kougka, and Ruben others Tous. Dynamic configuration of partitioning in spark applications. *IEEE Transactions on Parallel and Distributed Systems*, 28(7):1891–1904, 2017.

[38] Ayat Fekry, Lucian Carata, Thomas Pasquier, et al. To tune or not to tune? in search of optimal configurations for data analytics. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2494–2504, 2020.

[39] Yang Li, Huaijun Jiang, Yu Shen, et al. Towards general and efficient online tuning for spark. *Proceedings of the VLDB Endowment*, 16(12):3570–3583, 2023.

[40] Yu Shen, Xinyuyang Ren, Yupeng Lu, Huaijun Jiang, Huanyong Xu, Di Peng, Yang Li, Wentao Zhang, and Bin Cui. Rover: An online spark sql tuning service via generalized transfer learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4800–4812, 2023.

[41] Valentin Dalibard, Michael Schaarschmidt, and Eiko Yoneki. Boat: Building auto-tuners with structured bayesian optimization. In *Proceedings of the 26th International Conference on World Wide Web*, pages 479–488, 2017.

[42] Dan Li and Evangelos Kanoulas. Bayesian optimization for optimizing retrieval systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 360–368, 2018.

[43] Tirthak Patel and Devesh Tiwari. Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 193–206. IEEE, 2020.

[44] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. Satori: efficient and fair resource partitioning by sacrificing short-term benefits for long-term gains. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 292–305. IEEE, 2021.

[45] Tiannuo Yang, Ruobing Chen, Yusen Li, Xiaoguang Liu, and Gang Wang. Cotuner: A hierarchical learning framework for coordinately optimizing resource partitioning and parameter tuning. In *Proceedings of the 52nd International Conference on Parallel Processing*, pages 317–326, 2023.

[46] Chao Zhang, Shiwei Wu, Haoxin Zhang, Tong Xu, Yan Gao, Yao Hu, and Enhong Chen. Notellm: A retrievable large language model for note recommendation. In *Companion Proceedings of the ACM on Web Conference 2024*, pages 170–179, 2024.

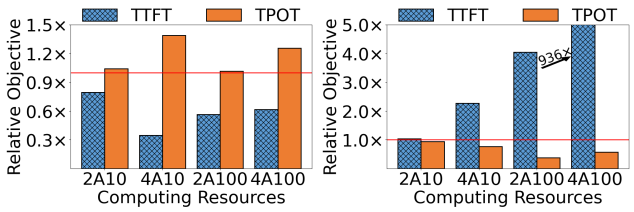
[47] Xubin Ren, Wei Wei, Lianghao Xia, Lixin Su, Suqi Cheng, Junfeng Wang, Dawei Yin, and Chao Huang. Representation learning with large language models for recommendation. In *Proceedings of the ACM on Web Conference 2024*, pages 3464–3475, 2024.

[48] Prateek Sancheti, Kamalakar Karlapalem, and Kavita Vemuri. Llm driven web profile extraction for identical names. In *Companion Proceedings of the ACM on Web Conference 2024*, pages 1616–1625, 2024.

[49] Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. Fine-tuning llama for multi-stage text retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2421–2425, 2024.

[50] Marc Stelmack, Nari Nakashima, and Stephen Batill. Genetic algorithms for mixed discrete/continuous optimization in multidisciplinary design. In *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, page 4771, 1998.

[51] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.



(a) TTFT-only optimization. (b) TPOT-only optimization.

Figure 14: Conflicting TTFT and TOPT. Decreased TTFT often leads to increased TPOT, and vice versa. The red line indicates the objective value under the default parameter configuration.

A Appendix

A.1 Relevance to the Systems and Infrastructure for Web

Due to the powerful natural language understanding and generation capabilities of LLMs, they have been widely employed in web applications, including recommendation systems [46, 47], classification tools[48], and information retrieval [49] applications, as well as intelligent web chatbots such as ChatGPT[2] and Claude[3].

Many of these LLM-based web applications leverage LLM inference services provided by cloud vendors to deploy their LLMs and process requests through APIs. Consequently, LLM inference services have emerged as a critical component of the web's infrastructure. By tuning the parameters of LLM inference engines to optimize SLOs for LLM inference services, cloud vendors can boost the performance of LLM-based web applications during periods of heavy workload. This not only enhances customer satisfaction to strengthen the competitive edge of cloud vendors in the market but also sustains the advancement of the intelligent web.

Therefore, we believe that this study closely aligns with the systems and infrastructure for web, particularly in relation to machine learning (ML) and artificial intelligence (AI) systems for web.

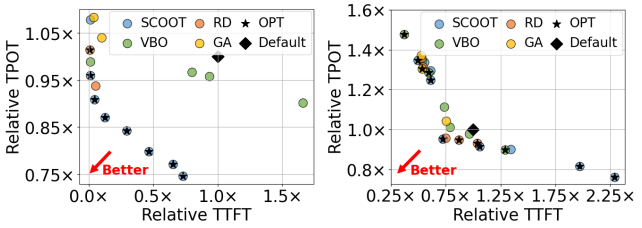
A.2 Conflicting TTFT and TPOT

We separately take TTFT and TPOT as the only optimization objectives and conduct grid searches for inference services of LLAMA2-13B deployed on various computing resources under the BOT request trace. Experimental results are shown in Fig. 14, which confirms that for TTFT and TPOT, optimizing for one always compromised the other. The reason is that, to decrease TPOT, it is significant to decrease the batch size controlled by the parameter `max-num-seqs` to reduce the per-token latency, which causes a long queuing time for requests, thus leading to a large TTFT. Conversely, to reduce TTFT, a large batch size is always set to allow newly arrived requests to be processed promptly without queuing for a long time. However, a large batch size often slows down inference, leading to increased TPOT.

A.3 Baseline Description

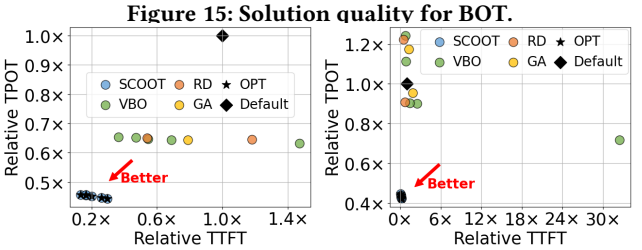
- **Random Sampling (RD)**: Sobol sequence-based Quasi-Monte Carlo [18] is utilized to uniformly sample configuration points from the search space.

[2]https://openai.com/chatgpt
[3]https://claude.ai



(a) Optimized TTFT & TPOT under LLAMA2-7B. (b) Optimized TTFT & TPOT under LLAMA2-13B.

Figure 15: Solution quality for BOT.



(a) Optimized TTFT & TPOT under LLAMA2-7B. (b) Optimized TTFT & TPOT under LLAMA2-13B.

Figure 16: Solution quality for SQL.

- **Genetic Algorithm (GA)**: Mixed variable GA [50] is used to resolve single-objective optimization problems while NSGA2 [51] is utilized to handle multi-objective situations. For GA, we assign a penalty objective value when constraints are violated and set the population size to 10.
- **Vanilla BO (VBO)**: VBO is implemented using HEBO [21]. It uses UCB and EHVI as acquisition functions to handle single-objective and multi-objective optimization situations, respectively. For VBO, we assign a penalty objective value when constraints are violated and use a random forest-based surrogate model to learn objective functions and constraints at the same time.

A.4 Solution Quality

Figures 15 and 16 plot the Pareto sets of configuration points sampled in the process of tuning the BOT and SQL applications under 4A10 for SCOOT and baselines. The plotted points represent the "locally optimal solutions" identified by each tuning method. Additionally, we mark the global Pareto set from these locally optimal configuration points with ★. From Fig. 15, we can observe that for BOT, the majority of globally optimal configuration points are contributed by SCOOT. Besides, SCOOT yields a wider range of its globally optimal points compared with baselines. From Fig. 16, we can find that for SQL, all the globally optimal configuration points are contributed by SCOOT. These experimental results confirm the superiority of SCOOT in both the optimality and diversity of SLO optimization in multi-objective situations.

SCOOT achieves a broader range of solutions because it uses EHVI as the acquisition function for multi-objective problems to optimize the Pareto front of TTFT and TPOT. Besides, since the default configuration is chosen as the reference point, EHVI ensures that the TTFT and TPOT of the configuration with maximal HV are both better than the default, achieving the goal of optimizing both objectives simultaneously. Since VBO also employs EHVI, its solution range is broad. However, because it cannot effectively handle constraints, the optimality of the solutions is compromised.