TimeBill: Time-Budgeted Inference for Large Language Models

Qi Fan, An Zou, Yehan Ma*

Shanghai Jiao Tong University
{fanqi666, an.zou, yehanma}@sjtu.edu.cn

Abstract

Large Language Models (LLMs) are increasingly deployed in time-critical systems, such as robotics, autonomous driving, embodied intelligence, and industrial automation, where generating accurate responses within a given time budget is crucial for decision-making, control, or safety-critical tasks. However, the auto-regressive generation process of LLMs makes it challenging to model and estimate the end-to-end execution time. Furthermore, existing efficient inference methods based on a fixed key-value (KV) cache eviction ratio struggle to adapt to varying tasks with diverse time budgets, where an improper eviction ratio may lead to incomplete inference or a drop in response performance. In this paper, we propose TimeBill, a novel time-budgeted inference framework for LLMs that balances the inference efficiency and response performance. To be more specific, we propose a fine-grained response length predictor (RLP) and an execution time estimator (ETE) to accurately predict the end-to-end execution time of LLMs. Following this, we develop a time-budgeted efficient inference approach that adaptively adjusts the KV cache eviction ratio based on execution time prediction and the given time budget. Finally, through extensive experiments, we demonstrate the advantages of TimeBill in improving task completion rate and maintaining response performance under various overrun strategies.

1 Introduction

With the development of Large Language Models (LLMs), LLMs have been widely applied in time-critical systems, such as robotics, autonomous driving, embodied intelligence, and industrial automation. For instance, Autoware.Flex (Song et al. 2024) utilizes LLMs to translate natural language instructions into a format that autonomous driving systems can understand during the driving process. DriveGPT4 (Xu et al. 2024) employs LLMs to perceive the driving environment and generate driving decisions along with corresponding explanations. In these scenarios, hard or firm deadlines may be introduced, especially when LLMs are involved in decision-making, control, or safety-critical tasks. The inference of LLMs should be completed within a specific time budget while ensuring the performance of the response. Therefore, it is essential to model the end-to-end execution time of LLM inference and balance between inference efficiency and response performance under the given time budget. However, applying LLMs in hard real-time systems faces numerous challenges.

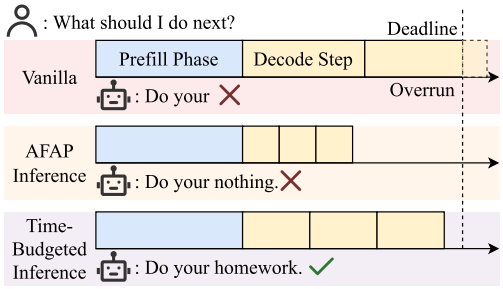

Figure 1: An example of different inference strategies. The vanilla inference may overrun and miss the deadline, resulting in incomplete output. The As-Fast-As-Possible (AFAP) strategy will degrade the response performance, while time-budgeted inference improves the response performance under timing constraints.

Unlike Convolutional Neural Networks (CNNs), LLMs exhibit significant uncertainty in the end-to-end execution time due to the auto-regressive generation process (Brown et al. 2020). Since the end-to-end execution time of LLMs is closely related to the number of generated tokens, namely response length, accurate modeling requires fine-grained prediction of response length. Several predictors rely on coarse-grained classification (Qiu et al. 2024; Jin et al. 2023), further increasing the error in modeling. Moreover, the response length is influenced by a series of factors such as input content and the LLM itself (Perez-Ramirez, Kostic, and Boman 2025). Different input content can lead to various response lengths, while even with the same input, different LLMs may generate responses with diverse lengths. Therefore, a fine-grained predictor well-aligned with the target LLM to be deployed is crucial for accurately modeling the end-to-end execution time of LLMs.

Existing efficient inference methods for LLMs can be categorized into two types: offline and online. Offline efficient inference methods, including quantization (Xiao et al. 2023; Lin et al. 2024; Frantar et al. 2023) and pruning (Frantar and Alistarh 2023; Ma, Fang, and Wang 2023), compress the

*Corresponding author

model before deployment to reduce resource consumption. However, they cannot adjust according to the time budget at runtime. Online methods are orthogonal to the offline ones, involving key-value (KV) cache eviction (Xiao et al. 2024; Li et al. 2024a; Xiao et al. 2025) and quantization (Hooper et al. 2024; Liu et al. 2024). Furthermore, the impact of KV cache eviction varies across different tasks (Li et al. 2024a), and different tasks may also have different time budgets. A fixed KV cache eviction ratio lacks the flexibility to handle such diverse scenarios. For instance, a higher ratio helps meet time budgets but harms response performance, while a lower ratio results in overruns. Therefore, it is vital to develop an efficient inference method that dynamically adjusts the KV cache eviction ratio based on the time budget, enabling LLMs to complete inference on time while maintaining response performance.

In this paper, we propose TimeBill, a time-budgeted inference framework for LLMs. As shown in Fig. 1, different from the As-Fast-As-Possible (AFAP) strategy, TimeBill balances inference efficiency and response performance. Beginning with presenting the problem formulation of time-budgeted inference for LLMs, we introduce a fine-grained response length predictor (RLP) and a workload-guided execution time estimator (ETE) to accurately predict the end-to-end execution time of LLMs. Furthermore, we develop a time-budgeted efficient inference method, which adapts the KV cache eviction ratio based on the execution time prediction and the time budget. Finally, we conduct a series of experiments and demonstrate the effectiveness of the TimeBill framework. The contributions are fourfold.

- We present the problem formulation of time-budgeted inference for LLMs and introduce a novel framework named TimeBill, which balances the timing performance of inference and response performance.
- We construct a fine-grained response length predictor (RLP), providing precise response length prediction of the target LLM to be deployed.
- We propose a workload-guided execution time estimator (ETE) with analytical modeling and profiling integrated, offering accurate end-to-end execution time estimation.
- We develop a time-budgeted efficient inference mechanism, which effectively adjusts the KV cache eviction ratio based on the execution time prediction and time budget, thereby improving task completion rate and maintaining response performance.

2 Related Work

2.1 Execution Time Estimation

Recently, real-time inference has been studied in deterministic DNNs, ensuring strict time constraints in time-critical applications. Chen et al. (Chen et al. 2024) map the fully-connected DNNs to a segmented task model on heterogeneous platforms. Kang et al. (Kang et al. 2021) establish response time analyses of layered DNNs and introduce quantization and runtime layer migration to reduce inference time. However, unlike the fixed-structure deterministic DNN, the end-to-end execution time of LLMs exhibits uncertainty due to the auto-regressive generation (Brown et al. 2020), which makes the execution time dependent on the dynamic response lengths.

A series of response length predictors have been proposed. For instance, PiA (Zheng et al. 2023) uses fine-tuning or prompt engineering to enable the target LLM to predict its response length before answering the question. ProxyModel (Qiu et al. 2024) builds a 5-class classifier based on BERT (Devlin et al. 2019) to predict which bucket the response length will fall into. S^3 (Jin et al. 2023) uses DistilBERT (Sanh et al. 2019) to construct a 10-class classifier. However, BERT-based predictors struggle with long input content. On the other hand, coarse-grained predictors cannot provide precise predictions for response time estimation. Therefore, a fine-grained response length predictor that can handle long input sequences is crucial to perform accurate response time prediction.

In addition, some works explore predicting execution time based on the execution characteristics of LLMs. For example, RLM-ML (Imai et al. 2024) and LLMStation (He et al. 2025) combine the roofline model with machine learning to predict execution time through data collection and optimization. BestServe (Hu et al. 2025) uses an adaptive roofline model for prediction. However, ML-based execution time prediction methods lack interpretability and are not friendly to online prediction.

2.2 Efficient LLM Inference

Due to the high computational resource usage and inference latency, LLMs face challenges in real-time applications. To this end, a series of efficient inference methods for LLMs have been proposed, which are mainly divided into two categories: offline and online methods.

Offline methods compress the model before deployment for lower resource consumption and time cost during inference. For example, SmoothQuant (Xiao et al. 2023) quantizes both weights and activations, while AWQ (Lin et al. 2024) and GPTQ (Frantar et al. 2023) perform weight-only quantization. In addition, SparseGPT (Frantar and Alistarh 2023) and LLM-Pruner (Ma, Fang, and Wang 2023) apply pruning to the model weights. However, when facing varying time costs due to the dynamic response time during runtime, these methods are unable to adjust according to the time budgets.

Online methods achieve efficient inference primarily through runtime eviction and quantization of the key-value (KV) cache. For instance, StreamingLLM (Xiao et al. 2024), SnapKV (Li et al. 2024a), and DuoAttention (Xiao et al. 2025) discard less important parts of the KV cache. Meanwhile, KVQuant (Hooper et al. 2024) and KIVI (Liu et al. 2024) quantize the KV cache to 4 bits or lower. Although online methods allow runtime adjustments, they overlook the time budgets, which may lead to overruns or inaccurate responses. Therefore, an efficient inference method that accounts for the time budget while maintaining the response performance is essential.

3 Overview

In this section, we present the problem formulation of time-budgeted inference for LLMs and our TimeBill framework.

3.1 Time-Budgeted Inference Problem for LLMs

The inference process of LLMs consists of two phases, namely the prefill phase and the decoding phase (Zhong et al. 2024). During the prefill phase, LLMs process the input prompt x and produce the first output token $\hat{y}_0$. LLMs perform autoregressive generation in the subsequent decoding phase, which comprises $N-1$ sequential decoding steps, as shown in Fig. 2. In each decoding step, LLMs generate a new output token $\hat{y}_i$ based on the previously generated tokens, where $i \in [1, N-1]$ is the index of the decoding step.

In time-critical systems with hard deadline constraints (hard real-time systems), inference that exceeds the time budget T is considered a system failure (Goh and Anderson 2024). Therefore, the goal of time-budgeted inference for LLMs is to optimize the response performance while ensuring that inference completes within the time budget. To be more specific, the time-budgeted inference for LLMs can be formulated as

$$\max_{\theta} \quad \mathcal{M}(\hat{\mathbf{y}}(\theta), \mathbf{y}) \tag{1a}$$

$$\text{s.t.} \quad t_{\text{e2e}}(x, \theta) \leqslant T \tag{1b}$$

$$N \leqslant N_{\max}. \tag{1c}$$

Given LLMs, the execution time and response performance of generating $\hat{y}_i$ are affected by a series of configuration factors θ in the decoding phase, such as the KV cache eviction ratio α. The objective in (1a) is to configure θ at runtime for each LLM inference in order to optimize the response performance metric $\mathcal{M}(\cdot)$ of generated response content $\hat{\mathbf{y}}(\theta) = (\hat{y}_0, \hat{y}_1, \ldots, \hat{y}_i, \ldots, \hat{y}_{N-1})$ compared with the ground truth $\mathbf{y} = (y_0, \ldots, y_{N-1})$. And the end-to-end execution time is t_{e2e} that should stay within the given time budget T, which is described as constraint (1b). The inference process completes when a termination token is generated or the maximum generation length $N_{\max}$ is reached, i.e., $N \leqslant N_{\max}$ in (1c).

3.2 TimeBill Framework

Challenges of the time-budgeted inference problem for LLMs described in Prob. (1) are:

Challenge 1 [Run-time execution time t_{e2e} Estimation]: *t_{e2e} largely depends on the response length N, which cannot be established until the inference is done. Therefore, the run-time response length prediction is the first challenge. In addition, given the response length prediction, how to accurately estimate t_{e2e} is another challenge.*

Challenge 2 [Run-time LLM configuration θ]: *How to map the execution time estimation and decoding phase configuration θ is the first challenge. Furthermore, how to configure the θ at runtime to optimize the response performance $\mathcal{M}(\cdot)$ within a certain time budget T is another challenge.*

We propose the TimeBill framework to address the above challenges. The overview of the TimeBill framework is presented in Fig. 2. ① The fine-grained RLP based on Small Language Model (SLM) is proposed to predict response lengths of the target LLM inference, which will be introduced in Sec. 4.1. ② Given the predicted response length, the workload-guided ETE is constructed by integrating FLOPs analysis and profiling to offer accurate end-to-end execution time estimation, which will be introduced in Sec. 4.2. ③ Given the time budget and estimated execution time for each LLM inference, we develop a time-budgeted efficient inference mechanism based on the KV cache eviction. The mechanism establishes the optimal KV cache eviction ratio to maximize the response performance $\mathcal{M}(\cdot)$ within the time budget, which will be described in Sec. 5.

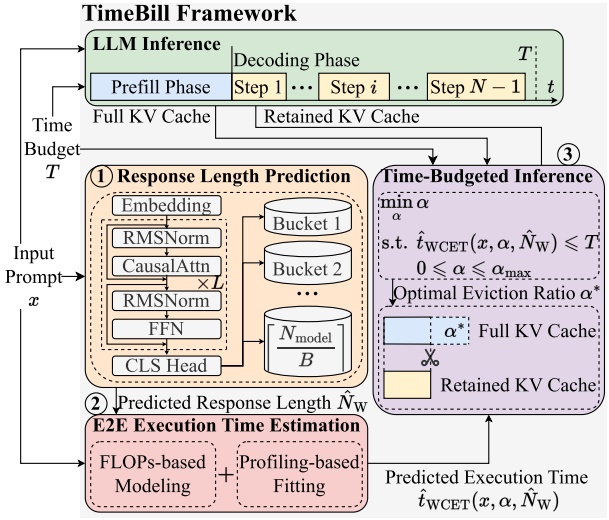

Figure 2: The overview of the TimeBill framework.

4 Fine-grained Execution Time Prediction for LLM

According to Challenge 1 mentioned in Sec. 3.2, the accurate end-to-end execution time prediction is the prerequisite for time-budgeted LLM inference. This section introduces the fine-grained RLP to predict the response length N, and the workload-guided ETE to estimate the end-to-end execution time t_{e2e}.

4.1 Fine-grained Response Length Predictor

Predictor Design To provide accurate end-to-end execution time estimation, we develop a fine-grained RLP. We define the response length prediction as a classification task instead of a regression task, since predicting the exact response length is challenging. Hence, the RLP needs to determine which bucket the response length will fall into, where the bucket size is fixed at B, as shown in Fig. 3. Due to the limited context length of BERT (Devlin et al. 2019), it is difficult to process longer inputs. Therefore, the RLP $\texttt{Predict}(\cdot)$ is based on a Small Language Model (SLM) to better process long input prompts, which has significantly fewer parameters than the target LLM. The architecture of RLP is shown in Fig. 3, which consists of an embedding layer, L decoder layers, and a classifica-

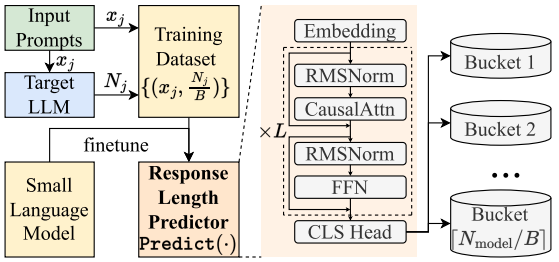

Figure 3: The overview of the proposed fine-grained response length predictor (RLP).

tion head. Each decoder layer includes four layers in sequence: an `RMSNorm` layer (Zhang and Sennrich 2019), a `CausalAttention` layer, another `RMSNorm` layer, and an `FFN with SwiGLU` layer (Shazeer 2020).

Since the response length largely depends on the target LLM (Perez-Ramirez, Kostic, and Boman 2025), we employ the knowledge distillation approach to better align the RLP with the target LLM. As shown in Fig. 3, for a given input prompt x_j, we collect the actual length N_j of the response generated by the target LLM, where j is the index of the data item. Therefore, we construct the training dataset consisting of pairs $(x_j, \lceil \frac{N_j}{B} \rceil)$, where $\lceil \frac{N_j}{B} \rceil$ represents the target bucket index, namely the classification label. According to the model card, the maximum generation capacity of the target LLM is N_{model} tokens, which exceeds the maximum generation length N_{max} specified during runtime, namely $N_{\text{model}} \geqslant N_{\text{max}}$. Therefore, there are $\lceil N_{\text{model}}/B \rceil$ buckets for classification during the training process.

According to the constraint (1c), we perform post-processing to limit the maximum predicted response length by N_{max}. Assuming the predicted bucket index is $\hat{n} = \texttt{Predict}(x)$, indicating the response length is between $(\hat{n}-1)B$ and $\hat{n}B$, and $\hat{n}B$ is reported as the predicted response length. The predicted response length $\hat{N}$ after post-processing can be obtained as follows:

$$\hat{N} = \min(N_{\text{max}}, \hat{n}B) = \min(N_{\text{max}}, \texttt{Predict}(x)B) \quad (2)$$

4.2 Workload-guided Execution Time Estimator

End-to-end execution time estimation is essential in hard real-time systems (Goh and Anderson 2024). Modeling the worst-case execution time (WCET) during system design ensures that each LLM inference meets deadlines. In this section, we develop a workload-guided ETE, integrating floating point operations (FLOPs) -based analytical modeling and profiling-based fitting.

FLOPs-based Modeling We adopt FLOPs-based modeling to analyze the relationship between computational workload and WCET, providing theoretical support for profiling-based fitting. Since most modern Transformer-based LLMs (Grattafiori et al. 2024; Yang et al. 2024) employ an architecture comprising an embedding layer, a series of decoder layers, and a language modeling head. The architecture of each decoder layer is `Norm-CausalAttention-Norm-FeedForward`. Given that matrix multiplications account for the majority of FLOPs (Narayanan et al. 2021), since there is no matrix multiplication in the embedding and `Norm` layer, we only analyze the FLOPs from the `CausalAttention` and `FeedForward` layers and the language modeling head `LMHead`. We suppose the number of FLOPs of each layer in the prefill phase and decoding step is denoted as $f^{\texttt{LayerName}}_{\text{prefill-phase}}$ and $f^{\texttt{LayerName}}_{\text{decoding-step}}$, respectively.

The main computation of the `CausalAttention` layer is $\text{CausalAttention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top \odot M}{\sqrt{d}}\right)V$, where d is the hidden size, $\odot$ denotes element-wise multiplication, and M is the causal attention mask (Vaswani et al. 2017). In the prefill phase, both Q and K have length N_x, where N_x is the length of the input prompt x. Therefore, $f^{\texttt{CausalAttention}}_{\text{prefill-phase}}$ is ***quadratic*** in N_x. In the decoding step, only the last generated token needs processing, so the length of Q is 1 and the length of K is N_{kv}, which denotes the current length of the KV cache. As a result, $f^{\texttt{CausalAttention}}_{\text{decoding-step}}$ is ***linear*** in N_{kv}. Similarly, the `FeedForward` layer handles the input prompt x consisting of N_x tokens during the prefill phase, and processes the last generated token in each decoding step. Since the `FeedForward` layer processes each input token independently (Yu et al. 2022), $f^{\texttt{FeedForward}}_{\text{prefill-phase}}$ is ***linear*** in N_x, while $f^{\texttt{FeedForward}}_{\text{decoding-step}}$ is ***constant*** and determined solely by the hyperparameters of the model (e.g., hidden size, intermediate size). The language modeling head `LMHead` takes the hidden state of the last input token as the input and produces the logits of the next token. Therefore, the FLOPs of the `LMHead` layer are solely related to the model architecture, regardless of the inference stage, namely $f^{\texttt{LMHead}}_{\text{prefill-phase}}$ and $f^{\texttt{LMHead}}_{\text{decoding-step}}$ are ***constant***.

Therefore, the number of FLOPs in the prefill phase $f_{\text{prefill-phase}}$ is quadratic in N_x with linear and constant terms included, while that in the decoding step $f_{\text{decoding-step}}$ is linear in N_{kv}, namely

$$f_{\text{stage}} = f^{\texttt{FeedForward}}_{\text{stage}} + f^{\texttt{CausalAttention}}_{\text{stage}} + f^{\texttt{LMHead}}_{\text{stage}}, \quad (3)$$

where stage is prefill phase or decoding step. Since the number of FLOPs and the corresponding execution time share the same form (Chowdhery et al. 2023), we can derive the estimated execution time of the prefill phase $\hat{t}_{\text{prefill-phase}}$ with respect to N_x (the length of input prompt x), and that of the i-th decoding step $\hat{t}^i_{\text{decoding-step}}$ with respect to N^i_{kv} as follows:

$$\hat{t}_{\text{prefill-phase}}(x) = aN_x^2 + bN_x + c \quad (4a)$$

$$\hat{t}^i_{\text{decoding-step}}(N^i_{\text{kv}}) = pN^i_{\text{kv}} + q, \quad (4b)$$

where a, b, c, p, and q are corresponding coefficients. In the next subsection, we will present a profiling-based and data-driven approach to establish these coefficients. Furthermore, we observe in Eq. (4b) that N^i_{kv} largely affects the execution time of each decoding step, the derivation and configuration of which will be discussed in detail.

Profiling-based Fitting Although FLOPs-based analytical modeling characterizes computational workload in terms of N_x and N^i_{kv}, actual execution time depends on implementation and hardware (Chowdhery et al. 2023). To this end, we propose a profiling-based fitting method to estimate the execution time. To be more specific, we establish coefficients

in Eq. (4) using data-driven approaches based on execution time profiling, given certain LLM models and hardware platforms. Taking the prefill phase as an example, the actual execution time $t_{\text{prefill-phase}}(N_x)$ for given N_x can be measured. Hence, a dataset with pairs $(N_x, t_{\text{prefill-phase}}(N_x))$ can be obtained to derive the coefficients a, b, and c in Eq. (4a). Well-established fitting methods, such as the Least Squares (LS), can be applied. The same applies to the decoding step.

End-to-end Execution Time Prediction According to Eq. (4b), the execution time of each decoding step depends on N_{kv}^i. Therefore, we explore the impact of the KV cache eviction ratio α on the execution time. Since KV cache eviction occurs after the prefill phase and before the decoding phase, a fraction α of the KV cache produced in the prefill phase will be evicted. As a result, in the i-th decoding step, the length of the KV cache is

$$N_{\text{kv}}^i(x, \alpha) = (1-\alpha)N_x + i - 1. \tag{5}$$

Given Eqs. (4b) and (5), the estimated execution time of the i-th decoding step becomes

$$\hat{t}_{\text{decoding-step}}^i(x, \alpha) = p((1-\alpha)N_x + i - 1) + q. \tag{6}$$

According to the predicted response length $\hat{N}$, which is obtained according to Eq. (2), the estimated execution time of the decoding phase is the sum of the execution time of all decoding steps, i.e.,

$$\hat{t}_{\text{decoding-phase}}(x, \alpha, \hat{N}) = \sum_{i=1}^{\hat{N}-1} \hat{t}_{\text{decoding-step}}^i(x, \alpha). \tag{7}$$

The end-to-end execution time prediction consists of the estimated execution time of the prefill and decoding phases. Thus, we estimate the execution time based on Eqs. (4a) and (7), i.e.,

$$\hat{t}_{\text{e2e}}(x, \alpha, \hat{N}) = \hat{t}_{\text{prefill-phase}}(x) + \hat{t}_{\text{decoding-phase}}(x, \alpha, \hat{N}). \tag{8}$$

Furthermore, considering WCET, we introduce the pessimistic factor k, $k \geqslant 1$, to $\hat{N}$. Hence, the pessimistic predicted response length becomes $k\hat{N}$. According to constraint (1c), the pessimistic predicted response length is $\hat{N}_{\text{W}} = \min(k\hat{N}, N_{\max})$. Based on Eq. (8), we can estimate the WCET of executing a LLM inference as

$$\hat{t}_{\text{WCET}}(x, \alpha, \hat{N}_{\text{W}}) = \hat{t}_{\text{prefill-phase}}(x) + \hat{t}_{\text{decoding-phase}}(x, \alpha, \hat{N}_{\text{W}}). \tag{9}$$

We will evaluate the impacts of k in the evaluation.

5 Time-Budgeted Efficient LLM Inference

This section presents the mechanism and corresponding system deployment for time-budgeted LLM inference.

5.1 Time-Budgeted LLM Inference Mechanism

According to Challenge 2 mentioned in Sec. 3.2, the configuration factors θ need to be adjusted at runtime due to the variances of input and response length. Therefore, we develop a time-budgeted efficient inference mechanism based on the KV cache eviction. Consequently, in the following analysis, we target the KV cache eviction ratio α as the configuration factor.

Since the response performance metric $\mathcal{M}(\cdot)$ is unknown until the inference is completed, which poses a challenge for maximizing the objective (1a) in Prob. (1). In general, the response performance is non-increasing as the KV cache eviction ratio α increases (Xiao et al. 2025). Thus, the objective function can be transformed from maximizing the response performance to minimizing the KV cache eviction ratio α.

Additionally, according to the constraint (1b), the end-to-end execution time t_{e2e} should stay within the time budget T, which cannot be established until the inference is completed as well. To this end, we use the predicted worst-case execution time $\hat{t}_{\text{WCET}}$ in Eq. (9) as a conservative prediction of t_{e2e}. Additionally, the overall overhead of executing $\texttt{Predict}(\cdot)$ and $\hat{t}_{\text{WCET}}$ prediction cannot be ignored, which we denote as $t_{\texttt{Predict}}(x)$. Since $\hat{t}_{\text{WCET}}$ prediction is the prerequisite for establishing α, the $t_{\texttt{Predict}}(x)$ can be measured directly. Therefore, the constraint (1b) is transformed to $t_{\texttt{Predict}}(x) + \hat{t}_{\text{WCET}}(x, \alpha, \hat{N}_{\text{W}}) \leqslant T$.

Therefore, we can obtain the converted problem of time-budgeted LLM inference, i.e.,

$$\min_{\alpha} \quad \alpha \tag{10a}$$

$$\text{s.t.} \quad t_{\texttt{Predict}}(x) + \hat{t}_{\text{WCET}}(x, \alpha, \hat{N}_{\text{W}}) \leqslant T \tag{10b}$$

$$0 \leqslant \alpha \leqslant \alpha_{\max}. \tag{10c}$$

Since an excessively large α may degrade $\mathcal{M}(\cdot)$ significantly, the maximum eviction ratio $\alpha_{\max}$ is introduced in constraint (10c). Substituting Eqs. (6), (7), and (9) into Eq. (10b), we can derive the optimal α^* by solving Prob. (10).

$$\alpha^* = \min\left(\alpha_{\max}, 1 - \frac{T - \hat{t}_{\text{prefill-phase}}(x) - t_{\texttt{Predict}}(x)}{pN_x(\hat{N}_{\text{W}} - 1)} + \frac{\hat{N}_{\text{W}} - 2}{2pN_x} + \frac{q}{pN_x}\right), \tag{11}$$

where $\hat{t}_{\text{prefill-phase}}(x)$ is defined in Eq. (4a). To summarize, given the input prompt x and optional maximum generation length $N_{\max}$, the coefficients in Eq. (4), and the pessimistic factor k, the optimal KV cache eviction ratio α^* for the time-budgeted efficient inference can be established to optimize the response performance within the time budget T.

5.2 System Deployment

As shown in Fig. 4, for each LLM inference, upon the input prompt x arriving, N_x is known. Subsequently, an offline-trained RLP $\texttt{Predict}(\cdot)$ is utilized to predict $\hat{N}$ according

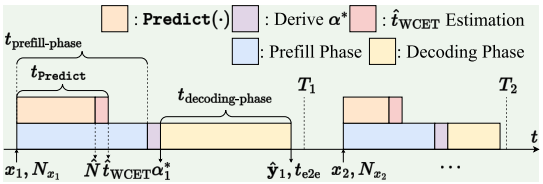

Figure 4: The timeline of TimeBill, where incoming arrows represent inputs (e.g., x_1, N_{x_1}) , and outgoing arrows represent outputs (e.g., $\hat{\mathbf{y}}_1, \alpha_1^*$).

to Eq. (2). Based on $\hat{N}$, k, and the offline-obtained coefficients in Eq. (4), $\hat{t}_{\text{WCET}}$ is estimated through Eq. (9). At the same time, the LLM inference begins the prefill phase, which takes $t_{\text{prefill-phase}}$. After the prefill phase, α^* can be determined through Eq. (11). α^* percent of the KV cache is evicted, and the retained KV caches are utilized during the subsequent decoding phase. After $t_{\text{decoding-phase}}$, the inference is completed, and the response $\hat{\mathbf{y}}$ is acquired.

Since $\hat{t}_{\text{WCET}}$ is the prerequisite for establishing α^*, $\hat{N}$ needs to be obtained before the decoding phase. Therefore, $\texttt{Predict}(\cdot)$ and $\hat{t}_{\text{WCET}}$ prediction can be executed in parallel with the prefill phase of the LLM on other processors, such as CPU or GPU. If $t_{\texttt{Predict}} \leqslant t_{\text{prefill-phase}}$, the negative impact of $t_{\texttt{Predict}}$ on response performance can be eliminated, i.e. $t_{\texttt{Predict}} = 0$ in Eq. (11). Therefore, in this work, we integrate our ETE in Sec. 4 with prompt compression (Li et al. 2024b). Note that the overhead of Eq. (9) is ignorable compared with $\texttt{Predict}(\cdot)$. Similar to Eq. (4a), the execution time of $\texttt{Predict}(\cdot)$ can be modeled as $\hat{t}_{\texttt{Predict}}(x_p) = a_p N_p^2 + b_p N_p + c_p$, where x_p is the input of the predictor and N_p is the length of x_p. Given $\hat{t}_{\text{prefill-phase}}$ obtained through Eq. (4a), an upper bound of N_p can be derived by solving $\hat{t}_{\texttt{Predict}}(x_p) \leqslant \hat{t}_{\text{prefill-phase}}(x)$. Accordingly, prompt compression is performed to compress the input prompt x into a shorter x_p.

Note that the time budget T can vary across the inferences, e.g., $T_1 \neq T_2$ in Fig. 4. Since $\hat{N}$, $\hat{t}_{\text{WCET}}$, and α^* are established for each inference, this can be handled naturally by TimeBill.

6 Evaluation

We first evaluate the efficacy of RLP in Sec. 6.2. And we demonstrate the performance of the ETE in Sec. 6.3. Then, we compare TimeBill with several state-of-the-art approaches in Sec. 6.4. The impact of the pessimistic factor is discussed in Sec. 6.5.

6.1 Experimental Setup

We utilize Qwen2.5-7B-Instruct (Yang et al. 2024) as the target LLM with a context length of 32,768 tokens and a maximum generation capacity $N_{\text{model}} = 8{,}192$ tokens. The test dataset is LongBench (Bai et al. 2024), and the response performance score is evaluated using the official evaluation metrics, such as F1 score, ROUGE-L (Lin 2004), and Levenshtein distance. The KV cache eviction is performed using SnapKV (Li et al. 2024a). The experiments are conducted on a server equipped with Intel(R) Xeon(R) Platinum 8350C and an NVIDIA A40 GPU.

TimeBill Implementation We employ Qwen2.5-0.5B-Instruct (Yang et al. 2024) as the SLM to build the proposed RLP $\texttt{Predict}(\cdot)$ with default 512 buckets (the number of buckets will be discussed in Sec. 6.2). To avoid training on the test set, the prompts from the Arena-Human-Preference-100k dataset (Tang, Chiang, and Angelopoulos 2025; Chiang et al. 2024) are used to construct the dataset for training the $\texttt{Predict}(\cdot)$. The execution time of the target LLM is profiled to build the ETE, where N_x and N_{kv} range from 0 to 32,768. The default pessimistic factor k is set to 5 (the value of k will be discussed in Sec. 6.5). The maximum KV cache eviction ratio α_{max} is set to 95%.

Approaches

- Vanilla inference (Vanilla), where the target LLM is directly used for inference.
- KV cache eviction with fixed α (Devoto et al. 2024; Zhang et al. 2023), where α is set to $25\%, 50\%, 75\%, 95\%$, respectively. We denote them as $\alpha = \text{x}\%$ in this section.
- Activation-aware Weight Quantization (AWQ), where the model weights is quantized to 4 bits (Lin et al. 2024).
- Our proposed **TimeBill**.

Overrun Strategies When an inference job is about to overrun and miss the deadline, the hard real-time system will apply an overrun strategy. We apply two of the most commonly used overrun strategies (Goh and Anderson 2024),

- *Kill*. The current job will be terminated and considered incomplete. The response is regarded as empty since it misses the deadline.
- *Skip-Next*. Skip the next few jobs until the current job completes. The skipped jobs are considered incomplete and do not any produce response.

The average response performance score across all data items in the test set and the completion rate are reported, where the completion rate is defined as the percentile of the number of completed jobs to the total number of jobs.

6.2 Efficacy of the Response Length Predictor

We compare our fine-grained RLP $\texttt{Predict}(\cdot)$ with BERT-based predictors, including ProxyModel (Qiu et al. 2024) and S^3 (Jin et al. 2023). We test the $\texttt{Predict}(\cdot)$ with different bucket sizes ($B = 16, 32, 64$), which correspond to 512, 256, and 128 buckets, respectively. In addition, we test $\texttt{Predict}(\cdot)$ modeled in a regression manner. We evaluate the prediction error using metrics including Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R-squared (R^2), where the ground-truth response lengths are collected using the target LLM. As shown in Tab. 1, $\texttt{Predict}(\cdot)$ outperforms ProxyModel and S3. The $\texttt{Predict}(\cdot)$ based on regression modeling performs poorly, indicating predicting exact response length directly is challenging. $\texttt{Predict}(\cdot)$ with 512 buckets achieves the best performance. As the number of buckets increases, the granularity of $\texttt{Predict}(\cdot)$ becomes finer, and the performance improves.

Methods	**Ours**				ProxyModel	S3
#Buckets	Reg	128	256	**512**	5	10
MAE	64.21	48.95	44.15	**42.71**	105.72	108.96
RMSE	103.30	87.57	78.63	**78.13**	136.79	148.91
R^2	0.516	0.652	0.719	**0.723**	0.152	-0.004

Table 1: The efficacy of different response length predictors.

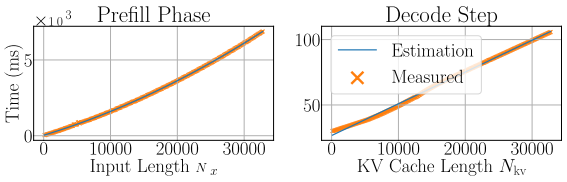

Figure 5: Fitted curves for estimating $\hat{t}_{\text{prefill-phase}}$, $\hat{t}_{\text{decoding-step}}$.

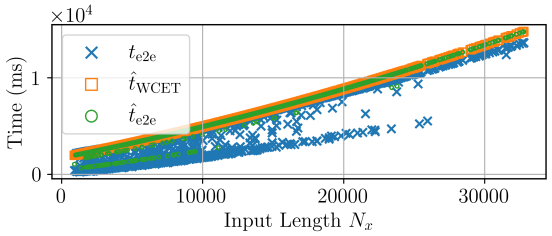

Figure 6: The performance of estimating $\hat{t}_{\text{e2e}}$ and $\hat{t}_{\text{WCET}}$.

6.3 Performance of the Execution Time Estimator

We first evaluate the performance of estimating the $\hat{t}_{\text{prefill-phase}}$ and $\hat{t}^{i}_{\text{decoding-step}}$ using Mean Absolute Percentage Error (MAPE). The MAPEs are 1.22% and 1.69% for the prefill phase and the decoding step, respectively, as shown in Fig. 5. Furthermore, we evaluate the performance of the end-to-end ETE, where `Predict`$(\cdot)$ with 512 buckets is utilized. The results of $\alpha = 0$ and $N_{\text{max}} = 64$ are presented in Fig. 6. We can see that $\hat{t}_{\text{e2e}}$ is close to the actual t_{e2e}, while $\hat{t}_{\text{WCET}}$ effectively provides an upper bound of t_{e2e}, demonstrating the effectiveness of the proposed ETE.

6.4 Comparison with Baselines

We conduct experiments on six different time budgets, ranging from 5 s to 10 s in one-second increments. The average response performance scores and completion rates under the *Kill* and *Skip-Next* strategies are shown in Fig. 7. We can see that TimeBill achieves the state-of-the-art performance in average score and maintains a competitive task completion rate. Since the Vanilla often exceeds the time budget, it performs poorly with a low task completion rate and average score. For KV cache eviction with fixed α, as α increases, the task completion rates increase, while the average score first increases and then decreases. This is because when α is small, the benefit gained from allowing more inferences to finish within T outweighs the loss in response performance, causing the average score to increase as α increases. However, when α is large, it degrades the response performance significantly. Furthermore, AWQ performs slightly better than the Vanilla. Moreover, TimeBill is orthogonal to it and can be effectively integrated with quantization. In contrast, TimeBill balances the inference efficiency and response performance, thereby achieving the highest average response performance scores among tested approaches while attaining a similar task completion rate as $\alpha = 95\%$.

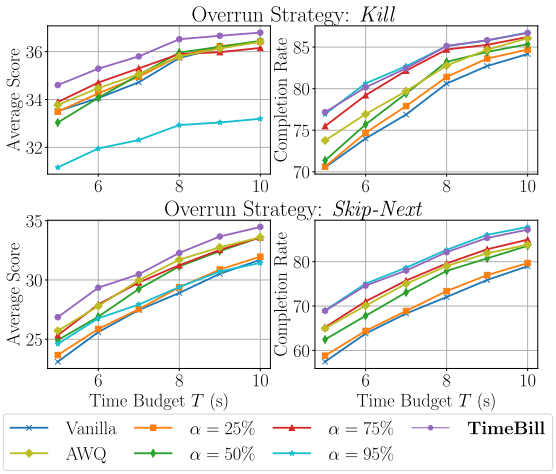

Figure 7: The average scores and completion rates of different approaches under *Kill* and *Skip-Next*.

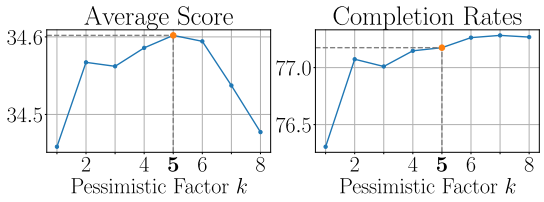

Figure 8: The average scores and completion rates with different pessimistic factors k under the overrun strategy *Kill*, where the time budget $T = 5$ s.

6.5 Impacts of the Pessimistic Factor k

We conduct experiments to demonstrate the impact of k under the *Kill* strategy, where $k \in [1, 8]$, and the time budget $T = 5$ s, as shown in Fig. 8. Since the more conservative $\hat{t}_{\text{WCET}}$ is, the higher the assurance of $t_{\text{e2e}} \leqslant \hat{t}_{\text{WCET}}$ (Vestal 2007), which is consistent with the fact that a pessimistic factor $k = 5$ is common in hard real-time systems (Joseph and Pandya 1986). When k is relatively small (1-5), increasing k results in a higher α and allowing more inferences to be completed within T. Thus, both the average score and the completion rate increase. However, a large k (6-8) leads to a large α, causing significant degradation in response performance and leading to a decrease in the average score. Therefore, k should be carefully selected.

7 Conclusion

In this paper, we propose TimeBill, a novel time-budgeted inference framework for LLMs. We present the problem formulation of time-budgeted inference for LLMs. We introduce a fine-grained RLP and a workload-guided ETE, enabling accurate end-to-end execution time prediction for LLMs. We develop a time-budgeted efficient inference method and provide the corresponding deployment. Through extensive experiments, we demonstrate the state-of-the-art performance of TimeBill in improving both response performance and completion rate.

Acknowledgments

This work is supported in part by the NSF of China under Grants 62473254 and 62202287, and in part by the Open Research Fund of Peng Cheng Laboratory under Grant 2025KF1B0010.

References

Bai, Y.; Lv, X.; Zhang, J.; Lyu, H.; Tang, J.; Huang, Z.; Du, Z.; Liu, X.; Zeng, A.; Hou, L.; et al. 2024. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. In *Annual Meeting of the Association for Computational Linguistics*, 3119–3137.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Chen, J.; Zou, A.; Xu, Y.; and Ma, Y. 2024. Scenic: Capability and scheduling co-design for intelligent controller on heterogeneous platforms. In *2024 IEEE Real-Time Systems Symposium (RTSS)*, 1–14. IEEE.

Chiang, W.-L.; Zheng, L.; Sheng, Y.; Angelopoulos, A. N.; Li, T.; Li, D.; Zhu, B.; Zhang, H.; Jordan, M.; Gonzalez, J. E.; et al. 2024. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*.

Chowdhery, A.; Narang, S.; Devlin, J.; Bosma, M.; Mishra, G.; Roberts, A.; Barham, P.; Chung, H. W.; Sutton, C.; Gehrmann, S.; et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240): 1–113.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 4171–4186.

Devoto, A.; Zhao, Y.; Scardapane, S.; and Minervini, P. 2024. A Simple and Effective L_2 Norm-Based Strategy for KV Cache Compression. *arXiv preprint arXiv:2406.11430*.

Frantar, E.; and Alistarh, D. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, 10323–10337. PMLR.

Frantar, E.; Ashkboos, S.; Hoefler, T.; and Alistarh, D. 2023. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In *International Conference on Learning Representations*.

Goh, J.; and Anderson, J. H. 2024. Towards Principled Budget Enforcement in Real-Time Systems. In *2024 IEEE Real-Time Systems Symposium (RTSS)*, 256–266. IEEE.

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

He, Y.; Yang, H.; Lu, Y.; Klimovic, A.; and Alonso, G. 2025. Resource Multiplexing in Tuning and Serving Large Language Models. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, 1639–1655.

Hooper, C.; Kim, S.; Mohammadzadeh, H.; Mahoney, M. W.; Shao, Y. S.; Keutzer, K.; and Gholami, A. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37: 1270–1303.

Hu, X.; Zeng, T.; Yuan, X.; Song, L.; Zhang, G.; and He, B. 2025. BestServe: Serving Strategies with Optimal Goodput in Collocation and Disaggregation Architectures. *arXiv preprint arXiv:2506.05871*.

Imai, S.; Nakazawa, R.; Amaral, M.; Choochotkaew, S.; and Chiba, T. 2024. Predicting LLM Inference Latency: A Roofline-Driven ML Method. In *Annual Conference on Neural Information Processing Systems*.

Jin, Y.; Wu, C.-F.; Brooks, D.; and Wei, G.-Y. 2023. S^3: Increasing GPU Utilization during Generative Inference for Higher Throughput. *Advances in Neural Information Processing Systems*, 36: 18015–18027.

Joseph, M.; and Pandya, P. 1986. Finding response times in a real-time system. *The Computer Journal*, 29(5): 390–395.

Kang, W.; Lee, K.; Lee, J.; Shin, I.; and Chwa, H. S. 2021. Lalarand: Flexible layer-by-layer cpu/gpu scheduling for real-time dnn tasks. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, 329–341. IEEE.

Li, Y.; Huang, Y.; Yang, B.; Venkitesh, B.; Locatelli, A.; Ye, H.; Cai, T.; Lewis, P.; and Chen, D. 2024a. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37: 22947–22970.

Li, Z.; Liu, Y.; Su, Y.; and Collier, N. 2024b. Prompt compression for large language models: A survey. *arXiv preprint arXiv:2410.12388*.

Lin, C.-Y. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, 74–81.

Lin, J.; Tang, J.; Tang, H.; Yang, S.; Chen, W.-M.; Wang, W.-C.; Xiao, G.; Dang, X.; Gan, C.; and Han, S. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6: 87–100.

Liu, Z.; Yuan, J.; Jin, H.; Zhong, S.; Xu, Z.; Braverman, V.; Chen, B.; and Hu, X. 2024. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. In *International conference on machine learning*.

Ma, X.; Fang, G.; and Wang, X. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36: 21702–21720.

Narayanan, D.; Shoeybi, M.; Casper, J.; LeGresley, P.; Patwary, M.; Korthikanti, V.; Vainbrand, D.; Kashinkunti, P.; Bernauer, J.; Catanzaro, B.; et al. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 1–15.

Perez-Ramirez, D. F.; Kostic, D.; and Boman, M. 2025. CASTILLO: Characterizing Response Length Distributions of Large Language Models. *arXiv preprint arXiv:2505.16881*.

Qiu, H.; Mao, W.; Patke, A.; Cui, S.; Jha, S.; Wang, C.; Franke, H.; Kalbarczyk, Z. T.; Başar, T.; and Iyer, R. K. 2024. Efficient interactive llm serving with proxy model-based sequence length prediction. *arXiv preprint arXiv:2404.08509*.

Sanh, V.; Debut, L.; Chaumond, J.; and Wolf, T. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Shazeer, N. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Song, Z.; Lv, M.; Ren, T.; Xue, C. J.; Wu, J.-M.; and Guan, N. 2024. Autoware. Flex: Human-Instructed Dynamically Reconfigurable Autonomous Driving Systems. *arXiv preprint arXiv:2412.16265*.

Tang, K.; Chiang, W.-L.; and Angelopoulos, A. N. 2025. Arena Explorer: A Topic Modeling Pipeline for LLM Evals & Analytics.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Vestal, S. 2007. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th IEEE international real-time systems symposium (RTSS 2007)*, 239–243. IEEE.

Xiao, G.; Lin, J.; Seznec, M.; Wu, H.; Demouth, J.; and Han, S. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, 38087–38099. PMLR.

Xiao, G.; Tang, J.; Zuo, J.; Guo, J.; Yang, S.; Tang, H.; Fu, Y.; and Han, S. 2025. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. In *International Conference on Learning Representations*.

Xiao, G.; Tian, Y.; Chen, B.; Han, S.; and Lewis, M. 2024. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*.

Xu, Z.; Zhang, Y.; Xie, E.; Zhao, Z.; Guo, Y.; Wong, K.-Y. K.; Li, Z.; and Zhao, H. 2024. Drivegpt4: Interpretable end-to-end autonomous driving via large language model. *IEEE Robotics and Automation Letters*.

Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Wei, H.; et al. 2024. Qwen2. 5 Technical Report. *arXiv preprint arXiv:2412.15115*.

Yu, G.-I.; Jeong, J. S.; Kim, G.-W.; Kim, S.; and Chun, B.-G. 2022. Orca: A distributed serving system for Transformer-Based generative models. In *USENIX Symposium on Operating Systems Design and Implementation*, 521–538.

Zhang, B.; and Sennrich, R. 2019. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32.

Zhang, Z.; Sheng, Y.; Zhou, T.; Chen, T.; Zheng, L.; Cai, R.; Song, Z.; Tian, Y.; Ré, C.; Barrett, C.; et al. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36: 34661–34710.

Zheng, Z.; Ren, X.; Xue, F.; Luo, Y.; Jiang, X.; and You, Y. 2023. Response length perception and sequence scheduling: An llm-empowered llm inference pipeline. *Advances in Neural Information Processing Systems*, 36: 65517–65530.

Zhong, Y.; Liu, S.; Chen, J.; Hu, J.; Zhu, Y.; Liu, X.; Jin, X.; and Zhang, H. 2024. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *USENIX Symposium on Operating Systems Design and Implementation*, 193–210.