# Deferred prefill for throughput maximization in LLM inference

Moonmoon Mohanty
Gautham Bolar
Preetam Patil
Indian Institute of Science
Bengaluru, India

UmaMaheswari Devi
Felix George
Pratibha Moogi
IBM Research
Bengaluru, India

Parimal Parag
Indian Institute of Science
Bengaluru, India

## Abstract

Large language models (LLMs) have led to ground-breaking improvements in the capabilities of generative AI (Gen-AI) applications, leading to their increased adoption, which in turn is leading to increasing volumes of user requests at LLM inference deployments. The existing common implementations of LLM inference engines perform a new prefill every time there is a prompt departure. We analytically model the inference system for a fixed batch size with large rate of prompt arrivals and scheduling prefills after a fixed number of prompt departures. We characterize the throughput of the system as number of prompts departing per unit time for different thresholds. We observe that to maximize throughput, there exists an optimal threshold on the number of prompt departures. We verify this observation with vLLM experiments, and compare the optimal threshold predicted theoretically to the experimentally observed ones.

***CCS Concepts:*** • **Computing methodologies** → **Modeling and simulation**; • **Computer systems organization** → **Neural networks**.

***Keywords:*** LLM inference systems, prompt completion time, throughput maximization, scheduling

**ACM Reference Format:**
Moonmoon Mohanty, Gautham Bolar, Preetam Patil, UmaMaheswari Devi, Felix George, Pratibha Moogi, and Parimal Parag. 2025. Deferred prefill for throughput maximization in LLM inference. In *Proceedings of The 5th Workshop on Machine Learning and Systems (EuroMLSys'2025).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3721146.3721962

## 1 Introduction

Large language models (LLMs) based on transformer architectures have led to a significant breakthrough in NLP and AI systems, as witnessed by the capabilities of applications such as ChatGPT [17] and agentic LLMs [14]. The close-to-human performance of LLM systems is made possible by their high complexity and large size, hence training and deploying them in a resource-efficient manner is critical for lowering cost and energy consumption. During inferencing, serving the requests at high throughput and low latency is key for seamless and compelling user experiences. Thus, improving the efficiency of LLM inferencing has become an active area of research, with ongoing development of several LLM inference optimization mechanisms—both algorithmic and systems-based [2, 8, 10–12].

One major shortcoming in the LLM optimization space is the lack of formal analytical approaches to reason about their performance that can help in choosing the right operating points. This is not surprising, given their highly complex and multi-dimensional nature and stochasticity in user requests and arrivals. Most optimization frameworks rely on extensive benchmarking and interpolating/extrapolating the benchmarked results via regression or ML approaches [1, 9].

At a high level, LLM inferencing consists of two main phases, the *prefill* phase and the *decode* phase [12, 16]. In the prefill phase, the input prompt received with a request is broken down into tokens and embedded into a vector space, for which Key-Value (KV) vector pairs and finally the first output token are generated using self-attention mechanism. The decode phase is iterative, with each iteration generating the next output token in an auto-regressive manner using the KV vectors of the input and preceding output tokens. Hence, the KV vectors generated for the input and output tokens are stored in a KV cache to avoid recomputing them. Since KV vectors for a token are independent of the other tokens, KV computation for an input prompt is parallelizable across the tokens of the prompt, making the prefill phase compute-intensive. On the other hand, due to its auto-regressive nature, output token generation is dependent on the KV vectors of all the preceding tokens and hence is sequential. To improve the utilization of the accelerator (GPU) and the throughput of the decode phase, multiple requests are concurrently decoded in a batch with all their KV vectors

*EuroMLSys'2025, Rotterdam, The Netherlands*

ACM ISBN 979-8-4007-1538-9/25/03
https://doi.org/10.1145/3721146.3721962

held in the KV cache. The number of concurrently decoded requests is referred to as the *batch size* of the inferencing server. Several variants of *LLM batching*, ranging from *static batching* to *continuous dynamic batching*, have been implemented to improve the overall throughput and per-request latency of LLM inferencing [2, 15].

In static batching, requests are served in statically grouped batches, each with a pre-specified number of requests, which are concurrently processed from the start to finish. Execution of a new batch commences only after all the requests in the previous batch complete. Static batching can lead to resource under-utilization due to the differences in the number of input and output tokens of requests and their asynchronous and non-deterministic arrivals. Dynamic, continuous batching attempts to overcome the shortcoming and improve system throughput by adding one or more new requests to a running batch at the end of a decode iteration depending on memory availability. When a new request is added, the decode phase is stalled to perform the prefill operations for it, which can increase the time to completion for the stalled requests. Thus, dynamic continuous batching offers a trade-off between throughput and latency by controlling when and how many new requests are admitted into a batch. Most inference servers, such as vLLM [8], TGIS [5], and ORCA [15] support dynamic continuous batching, admitting new requests after each iteration if KV cache memory permits. Some systems, such as TGIS, allow admitting new requests to be deferred if any request in the currently executing batch is nearing completion. However, there does not exist much guidance currently on how to determine the maximum batch size or when to admit new requests for a given batch size and desired throughput and latency bounds, except for the use of benchmarked results. In this paper, we take a first step to address this gap.

We attempt to maximize the throughput in terms of number of completed requests by deferring prefills to reduce stalls in the decode phase. It is easy to see that reducing decode stalls serves to lower the decode time. However, contrary to expectation, reducing stalls does not monotonically reduce throughput, but increases throughput up to a threshold, which we seek to analytically determine. For this, we model the needed components of an inference system, namely batching, KV cache, and the prefill and decode phases. The duration for which a prefill is deferred is in terms of the number of prompts that complete, termed the *departure threshold*. We show that in a backlogged system with infinite waiting requests, the decode batch reaches its maximum configured limit following a renewal process, using which we derive the optimal departure threshold. We validate the results via simulations and experiments using vLLM inference server and NVIDIA A100 GPU.

The main contributions of the work are as follows. We propose a departure threshold based scheduling algorithm for switching between the decode and prefill phases. We analytically model the inference system and find an expression for throughput under the proposed scheduling algorithm for large prompt arrival rates. We find an analytical approximation for throughput and find the optimal departure threshold that maximizes the approximate system throughput. We characterize a vLLM inference system to obtain the system parameters for our analytical model, and conduct simulations and experiments that verify our analytical insights.

## 2 System model

We model an inference system that can generate a maximum of $N$ tokens in parallel. There are three main components of the system that we model: (i) Characteristics of requests or prompts arriving into the system, (ii) KV cache where key-value pairs for requests are prefilled for output token generation, and (iii) prefill and decode times that govern the evolution of KV cache occupancy.

### 2.1 Requests

Each request arrives with a random number of input tokens and a random requirement of output tokens. For analytical tractability, we assume that for each request $n \in \mathbb{N}$, the number of input tokens $I_n$ is a constant $d_0$ and the required number of output tokens $J_n$ is independent and geometrically distributed with success probability $\alpha$. For simplicity, we consider a system with a large arrival rate of prompts such that the system always has an arbitrarily large number of prompts. This implies that there are always sufficient requests to completely fill the KV cache. We plan to relax the assumptions in the future by considering a realistic arrival process and token length distributions.

### 2.2 KV cache

We assume that the size of KV cache is fixed at $2Cd_1d_2$, i.e., it can store at maximum $C$ key-value pairs corresponding to $C$ prompts at any given time, where $d_1$ is the maximum sequence length allowed for each prompt and $d_2$ is the model dimension. Maximum allowed sequence length constrains the sum of input and output tokens to be $I_n + J_n \leqslant d_1$ for each prompt $n$. We assume that $d_1$ is sufficiently large such that $P\{J_n > d_1 - d_0\} = (1-\alpha)^{d_1 - d_0}$ is negligibly small. That is, for all practical purposes we can assume an infinitely supported geometric distribution for the number of output tokens. We denote the number of prompts in KV cache at the end of any time $t \in \mathbb{R}_+$ by $X_t$. By assumption, we have $X_t \leqslant C$ for all $t \in \mathbb{R}_+$. We define the history of the prompt occupancy process of KV cache until time $t$ as $\mathcal{F}_t \triangleq \sigma(X_s, s \leqslant t)$. The natural filtration of process $X$ is denoted $\mathcal{F}_\bullet \triangleq (\mathcal{F}_t : t \geqslant 0)$.

### 2.3 Prefill and decode

We consider a departure threshold based scheduling algorithm for switching between decode and prefill stages. For a departure threshold of $K$, the system switches from decode phase to prefill phase when there are $K$ or more prompt departures from the KV cache since last prefill. During the prefill phase, KV cache is prefilled to its maximum batch size of $C$ prompts from the infinitely backlogged requests. For each $k \in \mathbb{N}$, we define the $k$th instant when the KV cache is prefilled upto its maximum by $T_k$ and the $k$th instant when the KV cache has $K$ or more prompt departures by $S_k$. That is, we denote $k$th decode and prefill phases as

$$D_k \triangleq (T_{k-1}, S_k], \qquad\qquad P_k \triangleq (S_k, T_k].$$

Taking $X_0 \triangleq C$ and $T_0 \triangleq 0$, we observe that the beginning of $k$th decode phase is $T_k \triangleq \inf\{t > S_{k-1} : X_t = C\}$. Further, the beginning of $k$th prefill phase is

$$S_k \triangleq \inf\{t > T_k : X_t \leqslant C - K\}.$$

During the prefill phase, the system has an aggregate processing rate of $N$ parallel token computations in $t_p$ time units with an overhead of $c_p$ time units to switch from decode to prefill. During the $k$th prefill phase, the system needs to process $(C - X_{S_k})d_0$ input tokens and generate $(C - X_{S_k})$ output tokens. Since the inference system has a constant speed of $N/t_p$ token generations per unit time during prefill stage and there is a constant stall overhead of $c_p$, the end of $k$th prefill phase occurs at

$$T_k = S_k + c_p + \frac{t_p d_0}{N}(C - X_{S_k}). \tag{1}$$

During the decode phase, the system can generate at maximum $N \wedge X_t$ output tokens in parallel corresponding to $X_t$ prompts in the KV cache. Therefore, we focus on the case when $C \leqslant N$. We observe that decode time to generate $X_t$ tokens in parallel is $c_d + t_d X_t$, which is also called *time between tokens* (TBT). Assuming no contention between the $X_t$ prompts in a batch, $c_d$ can be considered to be the time to complete the model execution for one token generation per prompt in the GPU kernel.[1] However, it is known [2] that the decode phase is memory-bound and the $X_t$ prompts need to share the finite GPU memory bandwidth while accessing the KV-cache. As such $t_d$ can be considered to be the slowdown factor due to memory contention. During the $k$th decode phase, we can define the instant of $j$th parallel token generation time as $t_{k,j}$, where $t_{k,0} \triangleq T_{k-1}$. We denote the number of prompts in the KV cache at the instant $t_{k,j}$ as $X_{k,j} \triangleq X_{t_{k,j}}$. From the assumption on token generation times, we have $t_{k,j} = t_{k,j-1} + c_d + t_d X_{k,j-1}$. We define the number of parallel token generations before $K$ or more departures in the $k$th decode phase as $N_k \triangleq \left\{j \in \mathbb{N} : X_{k,j} \leqslant C - K\right\}$. We observe that $S_k = t_{k,N_k}$, and can write the length of the $k$th decode phase as

$$S_k - T_{k-1} = t_{k,N_k} - t_{k,0} = N_k c_d + t_d \sum_{j=0}^{N_k - 1} X_{k,j}. \tag{2}$$

We summarize the timing diagram of events in one decode and prefill phase in Figure 1.

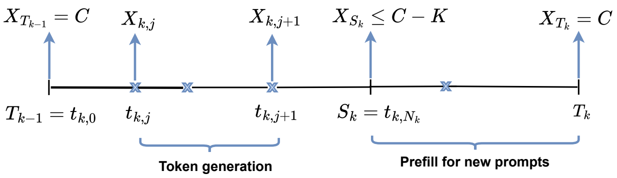


**Figure 1.** Timing diagram and state evolution between two prefills.

Since we are considering a system with large prompt arrival rate, we measure the system performance in terms of system throughput. The rate of change of number of prompts is nonpositive in decode phase $D_k$ and nonnegative in prefill phase $P_k$ for each $k \in \mathbb{N}$. The rate of prompt departures equals the negative rate of change of number of prompts, and thus the limiting throughput of the system as a function of departure threshold $K$ is $\rho(K) \triangleq \lim_{t\to\infty} \frac{1}{t}\int_{s \leqslant t}(-dX_s \vee 0)$.

**Problem 1.** We are interested in finding the optimal departure threshold of prompts for a prefill event, that maximizes the throughput of the inference system under consideration. That is, we wish to find $K^* \triangleq \arg\max\{\rho(K) : K \in [C]\}$.

## 3 Analysis

In this section, we compute the throughput in terms of ratio of $C - \mathbb{E}X_{S_k}$ and $\mathbb{E}(T_k - T_{k-1})$. This is difficult to compute, and optimal threshold can only be found numerically. Nevertheless, we propose an approximate way to compute throughput. We show that there exists an optimal departure threshold $K^*$ that maximizes the approximate throughput.

**Lemma 1.** *If prompt service times are geometrically distributed with success probability $\alpha$, then the evolution of $X_{k,j}$ is Markov for $j \in \{1, \dots, N_k\}$. In particular, the distribution of $X_{k,j}$ is a Binomial with parameters $(X_{k,j-1}, 1 - \alpha)$.*

*Proof.* From memoryless property of geometric distribution, the remaining output token distribution remains geometric with success probability $\alpha$. A prompt $n$ departs iff $J_n = 1$ and stays if $J_n > 1$, i.e. $\mathbb{1}_{\{J_n > 1\}}$ is the indicator of an existing prompt $n$ from time $t_{k,j-1}$ to remain in the KV cache at time $t_{k,j}$. Since $\mathbb{1}_{\{J_n > 1\}}$ is a Bernoulli random variable with mean $1 - \alpha$, the result follows from the facts that the number of output tokens for all prompts are *i.i.d.* and there are $X_{k,j-1}$ prompts at time $t_{k,j-1}$. □

**Lemma 2.** *The mean length of $k$th decode phase is*

$$\mathbb{E}(S_k - T_{k-1}) = c_d \mathbb{E}N_k + \frac{t_d}{\alpha}(C - \mathbb{E}X_{S_k}).$$

*Proof.* We define a filtration as $\mathcal{G}^k_\bullet \triangleq (\mathcal{G}^k_j : j \in \mathbb{N})$ where $\mathcal{G}^k_j = \sigma(X_{k,0}, \dots, X_{k,j})$. We observe that $N_k$ is a stopping

[1] plus a small vLLM engine overhead per decode iteration.

time adapted to $\mathcal{G}^k$ and we can write the following telescopic sum $C - X_{k,N_k} = \sum_{j\in\mathbb{N}}(X_{k,j-1} - X_{k,j})\mathbb{1}_{\{j\leqslant N_k\}}$. Taking expectation on both sides of the above equation, using the fact that $X_{k,j-1} - X_{k,j} \geqslant 0$ for $j \leqslant N_k$, and exchanging expectation and sum using monotone convergence theorem, we obtain $C - \mathbb{E}X_{k,N_k} = \sum_{j\in\mathbb{N}}\mathbb{E}[(X_{k,j-1} - X_{k,j})\mathbb{1}_{\{j\leqslant N_k\}}]$. Using the tower property of conditional expectation, the fact that $\{N_k \geqslant j\} \in \mathcal{G}^k_{j-1}$ and $X_{k,j-1}$ is $\mathcal{G}^k_{j-1}$ measurable, and monotone convergence theorem to exchange summation and expectation, we obtain

$$C - \mathbb{E}X_{k,N_k} = \mathbb{E}\sum_{j\in\mathbb{N}}\mathbb{1}_{\{j\leqslant N_k\}}(X_{k,j-1} - \mathbb{E}[X_{k,j} \mid \mathcal{G}^k_{j-1}])$$

The result follows from Lemma 1 and substituting in (2). □

**Theorem 1.** *For the inference system under consideration with prefill after K or more departures, the limiting throughput is*

$$\rho(K) = \frac{C - \mathbb{E}X_{S_1}}{\mathbb{E}T_1} = \Big(\frac{c_p + c_d\mathbb{E}N_1}{C - \mathbb{E}X_{S_1}} + \frac{t_d}{\alpha} + \frac{t_p d_0}{N}\Big)^{-1}.$$

*Proof.* We observe that $T_k, S_k$ are almost surely finite stopping times adapted to $\mathcal{F}_\bullet$, and $X_{T_k} = C$ for all $k \in \mathbb{Z}_+$. Further at each instant $T_k$, we have $C$ geometrically distributed output tokens with constant token generation times. From the memoryless property of geometric distributions independent of $\mathcal{F}_{T_k}$, it follows that $S_k - T_{k-1}$ is independent of $\mathcal{F}_{T_{k-1}}$ and identically distributed for all $k \in \mathbb{N}$. Further, we have $X_{S_k}$ is independent of $\mathcal{F}_{T_{k-1}}$ and identically distributed for all $k \in \mathbb{N}$. It follows that $(T_k : k \in \mathbb{Z}_+)$ are renewal instants.

We can consider the number of departures in $k$th renewal interval as the aggregate reward. Since $dX_s \leqslant 0$ for all $s \in D_k$ and $dX_s > 0$ for all $t \in P_k$, we obtain

$$\int_{T_{k-1}}^{T_k}(-dX_t \vee 0) = -\int_{T_{k-1}}^{S_k} dX_t = C - X_{S_k}.$$

The result follows from the application of renewal reward theorem [7] and Eq. (1) for prefill duration $T_k - S_k$. □

*Remark* 1. Computation of $\mathbb{E}X_{S_1}$ and $\mathbb{E}N_1$ are difficult analytically, and can only be computed numerically. However, we propose an approximate way of computing them. By definition, we have $C - X_{S_1} \geqslant K$, and hence $C - \mathbb{E}X_{S_1} \geqslant K$. Thus, we have $\rho(K)^{-1} \leqslant (c_p + c_d\mathbb{E}N_1)/K + t_d/\alpha + t_p d_0/N$. We approximate $X_{S_k} \approx C - K$ and use the lower bound on throughput as its approximation.

**Proposition 1.** *For small departure threshold K and large batch size C, the inverse of throughput for the inference system under large arrival rates, departure threshold based switching between phases, and geometrically distributed output token, can be approximated as*

$$\rho(K)^{-1} \approx \frac{1}{K}\Big(c_p + c_d\frac{\ln(1-\frac{K}{C})}{\ln(1-\alpha)}\Big) + \frac{t_d}{\alpha} + \frac{t_p d_0}{N}.$$

*Proof.* We approximate $X_{S_k} \approx C - K$ and thus from Theorem 1 and Lemma 2 it suffices to approximate $\mathbb{E}N_1$. We know that a binomial random variable with parameters $(N, p)$ concentrates around its mean $Np$ for large $N$. Hence, for small $K$ and large $C$, we can approximate the number of departures in iterations $j \leqslant N_1$ by $X_{1,j-1} - X_{1,j} \approx X_{k,j-1}(1-\alpha)$. We can approximate $X_{k,j} \approx X_{k,0}(1-\alpha)^j$ for $j \in [N_1]$ where $X_{k,0} = C$. Thus, we can approximate $C(1-\alpha)^{N_1} \approx C - K$ and thus $N_1 \approx \ln(1-\frac{K}{C})/\ln(1-\alpha)$. The result follows from substituting approximations for $X_{S_k}$ and $\mathbb{E}N_k$ in Theorem 1. □

*Remark* 2. We observe that the inverse throughput is positive and convex for $K \in [0, C]$ and hence has a unique minimum, indicating a unique departure threshold $K^*$ that maximizes the approximate throughput. We also note that $K^* > 1$ only if prefill stall time $c_p > 0$. The sequential time $t_d$ for each decode and prefill generation time per token $t_p d_0/N$ do not affect the shape of throughput. They merely reduce the throughput by a constant factor.

# 4 vLLM characterization

We perform empirical characterization on a vLLM inference serving system equipped with an accelerator (GPU) to investigate the correlation of prefill time and time-between-tokens with the departure threshold $K$. Specifically, we derive the system parameters $(c_p, t_p)$ for the prefill times and $(c_d, t_d)$ for decode times, to obtain the optimal solution to Problem 1 using the empirical observations from the vLLM server.

## 4.1 System setup

We use a virtual machine with eight dedicated cores (16 vCPUs) of an AMD EPYC 7513 processor and 64GB dedicated RAM. The VM has a dedicated NVIDIA A100 PCIe GPU (80GB VRAM) mapped through PCIe pass-through. For consistency and reproducibility, the GPU SM frequency and memory frequency are set to 1380 MHz and 1512 MHz, respectively. The GPU driver version is nvidia-550.127.08, and the CUDA library version is 12.4. We use pre-built binaries (version 0.6.3.post2) supplied by vLLM repository [6].

For characterization, we conduct inference on LLaMA [13], a 8.03B parameter model, and IBM-Granite [4], a 8.17B parameter model, using vLLM inference server. We set the maximum batch size for the Granite model to $C = 331$ and for LLama3 model to $C = 410$. For all the experiments, models were served using the OpenAI API compatible server endpoint of vLLM. We perform characterization for two datasets. The first one is a synthetic dataset, where we synthesize prompts that consist of a constant number of input tokens, i.e., $d_0 = 285$. The number of output tokens is sampled from a geometric distribution with mean $1/\alpha = 201$. These values are chosen to match the mean values from a real-life dataset which is our second dataset ShareGPT [3]. To analyze our proposed policy in a more realistic evaluation framework, we chose ShareGPT workload, which is a collection of user chats at chatGPT with varying sequence lengths and non-geometric distribution. We note that it is a filtered version of the original ShareGPT dataset, where the prompts with invalid responses are removed.

Apart from inference throughput, which we analytically compute, we also measure average and normalized prompt completion time. Throughput is computed as the average number of prompts completed per unit of time. The normalized prompt completion time is computed by dividing each prompt's completion time by the total number of output tokens. We note that the prompt completion time does not include the time spent in the input queue, as we analyze a backlogged queue system.

### 4.2 Proposed Scheduler implementation

The default vLLM scheduler performs prefill after every iteration (token generation during the decode phase) if there are prompts in the queue and there is space in KV-cache (due to the departure of a prompt). Contrastingly, the proposed departure threshold based scheduling algorithm performs prefills only after accumulating a threshold of $K$ or more departures. We achieve this by modifying the `_schedule_default()` method within `vllm/core/scheduler.py`. To keep the inference server's queue backlogged, we configure the client to send an initial burst of requests that is much larger than the batch size $C$ configured at the server. Completed prompts at the server are immediately replaced by new request prompts from the client to keep the server queue backlogged.

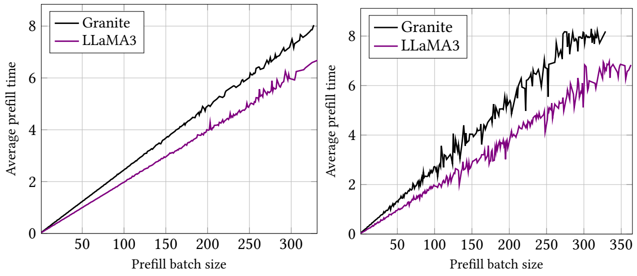
### 4.3 Measurements

We make measurements of prefill and decode times by logging timestamps of the scheduling and processing events occurring within the core inference engine. On invocation of the scheduler, we log its decision (prefill/decode), batch size, and time stamp. We also log the time stamp when the worker (GPU kernel) returns the tokens for the current batch.

### 4.4 Results

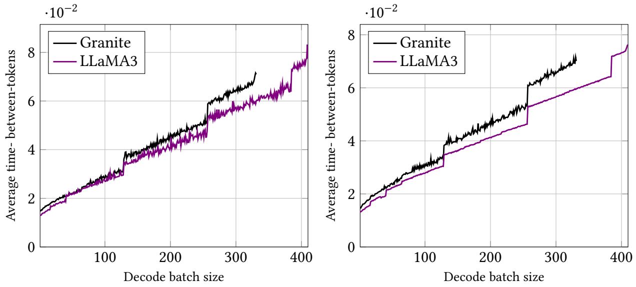
Next, we present the results of the characterization experiment. In Figure 2, we plot the average prefill times obtained for Granite and LLaMA3 for both the synthetic and ShareGPT datasets. We observe that the prefill time is an affine function of the batch size, with two components: (a) a scheduling overhead $c_p$ owing to the transfer of prompts from the input queue to the vLLM engine plus a fixed overhead in switching from decode to prefill, and (b) the time to process the input tokens corresponding to the prefill batch size. It is known [2, 8] that prefill is a compute-heavy phase, leading to prefill time increasing linearly with batch size beyond the GPU computation capacity.

Similarly, in Figure 3, we present the mean TBT obtained for both models and data sets. Similar to the prefill time, the mean TBT is an affine function of the decode batch size. In accordance with the system model for decode time in Section 2, the y-axis intercept is $c_d$, the interference-free compute time, while the slope of the curve $t_d$, represents memory contention slowdown factor. We also observe that the mean TBT curve exhibits sharp jumps at specific batch sizes.



**(a)** Synthetic dataset **(b)** ShareGPT dataset

**Figure 2.** Average prefill time ($s$) vs. prefill batch size.



**(a)** Synthetic data **(b)** ShareGPT dataset

**Figure 3.** Average TBT ($s$) vs. decode batch size.

## 5 Evaluation results

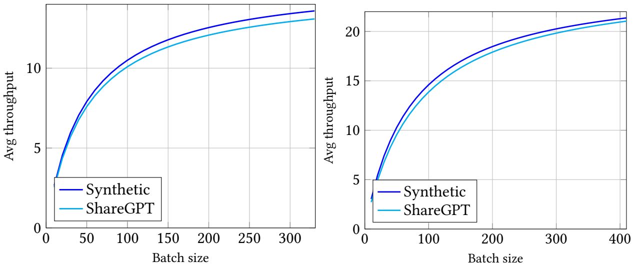
We validate the efficacy of the proposed departure threshold-based scheduling policy by measuring the impact of the departure threshold $K$ on crucial performance metrics. We compare the analytical results with the model simulations and experiments. We obtain the system parameters for the analytical result by curve-fitting the characterization results in Section 4, which are utilized for the model simulations and the numerical evaluation of analytical results. The experimental setup is similar to that used for empirical characterization in Section 4. Figure 4 depicts the behavior of average throughput with varying maximum KV cache batch size $C$. As expected from Proposition 1, the average throughput is an increasing function of the batch size.

We present the average of throughput and prompt completion time ($s$) for the Granite model in Figure 5 and for the LLaMA3 model in Figure 6, for ShareGPT dataset. Proposition 1 suggests that there exists an optimal departure threshold $K^*$ corresponding to the maximum throughput for a fixed maximum batch size. We observe this in Figure 5a and Figure 6a. The theoretically predicted optimal throughput

**Table 1.** Optimal performance metrics

| **Model** | **Proposed policy** | | | | **Default policy *K* = *1*** | | |
|---|---|---|---|---|---|---|---|
| **Type** | ***Tpt(Exp), K*** | ***Tpt(Num), K*** | ***Opt mean time**** | ***Opt 95% time**** | ***Tpt(Exp)*** | ***Mean time**** | ***95% time**** |
| Granite (shGPT) | (12.92, 32) | (13.09, 26) | 0.1161 | 0.1656 | 11.38 | 0.1230 | 0.1400 |
| LLaMA3 (shGPT) | (17.24, 75) | (21.04, 58) | 0.1289 | 0.210625 | 15.16 | 0.1297 | 0.149398 |

*Normalized



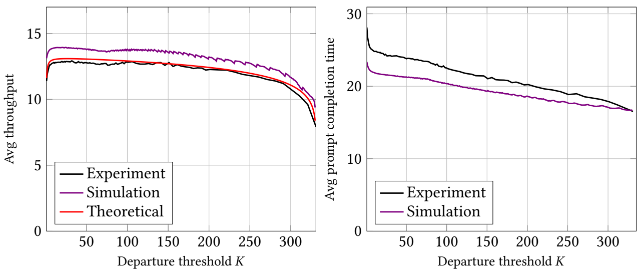
**(a)** Granite model **(b)** Llama3 model

**Figure 4.** Throughput vs. batch size.

is within 1.3% of the experimental optima for Granite, and 18% of the experimental optima for Llama3. This shift in optima arises due to single parameter $c_d$ fitting the characterization curve which appears to have different values in different regime. For Granite model, we notice that our policy achieves an improvement of ∼ 13.5% in throughput, accompanied by a reduction of ∼ 14% in average prompt completion time, as seen in Figure 5, as compared to the default scheduling policy. We observe a significant improvement over the existing policy for Llama3 model as well, with ∼ 13.7% higher throughput along with a ∼ 17% lower average prompt completion time, as seen in Figure 6. The exact values of the optimal metrics are tabulated in Table 1. In addition, we also look into the normalized prompt completion times for both the models. Figure 7 shows the variance of average normalized prompt completion time and the 95%-ile prompt completion time on the departure threshold $K$. It is expected to increase with throughput. We observe that the normalized 95%-ile prompt completion time is within 1.5× the normalized mean completion time.

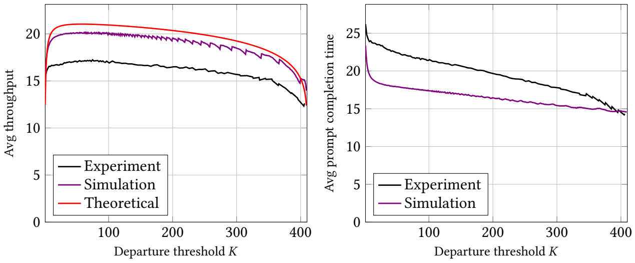
## 6 Conclusion and future work

We analytically modeled an inference system for large request arrival rates and proposed a departure threshold based scheduling policy for switching between decode and prefill phases. We predicted the existence of a non-trivial optimal departure threshold $K^*$ that maximizes the system throughput. Experimental results corroborate the prediction of the existence of an optimal threshold $K^*$. The analytically predicted threshold $K^*$ was close to the experimentally obtained one for Granite, verifying the goodness of the model.



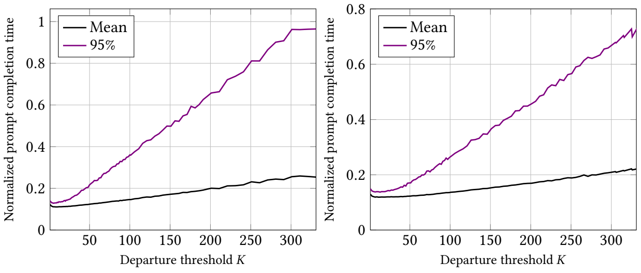
**(a)** Throughput **(b)** Avg prompt completion time

**Figure 5.** Performance metrics vs. departure threshold $K$ for the Granite model with the ShareGPT dataset



**(a)** Throughput **(b)** Avg prompt completion time

**Figure 6.** Performance metrics vs. departure threshold $K$ for the LLaMA3 model with the ShareGPT dataset



**(a)** Granite model **(b)** Llama3 model

**Figure 7.** Normalized prompt completion time ($s$/token) as a function of number of departures required for new prefill, for ShareGPT dataset

We observed that our proposed policy, in comparison with the default policy, achieves at least a 13% improvement in throughput and at least a 14% reduction in average prompt completion time. Our proposed policy can be incorporated into the inference engine offline by computing the optimal departure threshold and adjusting the parameters of the inference engine accordingly or online by accessing the traffic characteristics and deciding the parameters in real time. An interesting future direction is to model tensor parallelism for handling larger models through multi-GPU configuration.

## Acknowledgments

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions.

This research was supported in part by IBM Research India under IBM-IISc Hybrid Cloud Lab (IIHCL) open research collaboration; in part by Qualcomm Inc. under Qualcomm University Relations 6G India; in part by Anusandhan National Research Foundation (ANRF) Grant CRG/2023/008854; in part by UK-India Education and Research Initiative (UKIERI) Grant SPARC/2024/3927; and in part by Centre for Networked Intelligence (a Cisco Corporate Social Responsibility (CSR) Initiative) at Indian Institute of Science, Bengaluru. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

## References

[1] Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav Gulavani, Ramachandran Ramjee, and Alexey Tumanov. 2024. VIDUR: A Large-Scale Simulation Framework for LLM Inference. In *Mach. Learning Sys. (MLSys)* (Santa Clara, CA, USA), Vol. 6. ACM, USA, 351–366.

[2] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *USENIX Symp. Op. Sys. Des. Impl. (OSDI)* (Santa Clara, CA, USA), Vol. 18. ACM, USA, 117–134.

[3] Anonymous. 2023. ShareGPT Vicuna Unfiltered Dataset. Hugging Face Datasets. https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered Accessed: 2025-02-05.

[4] Anonymous. 2024. *Granite 3.1 Language Models*. IBM. https://github.com/ibm-granite/granite-3.1-language-models/ Accessed: 2025-02-05.

[5] Anonymous. 2024. *Text Generation Inference*. HuggingFace. https://github.com/huggingface/text-generation-inference

[6] Anonymous. 2025. vLLM: A high-throughput and memory-efficient inference and serving engine for LLMs. https://github.com/vllm-project/vllm

[7] Erhan Çinlar. 1975. Exceptional Paper – Markov Renewal Theory: A Survey. *Mgmt. Sci.* 21, 7 (March 1975), 727–752.

[8] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with Paged Attention. In *Sym. Op. Sys. Princ. (SOSP)* (Koblenz, Germany), Vol. 29. ACM, USA, 611–626.

[9] Malgorzata Lazuka, Andreea Anghel, and Thomas Parnell. 2024. LLM-Pilot: Characterize and Optimize Performance of your LLM Inference Services. In *Inter. Conf. High Perf. Comput., Netw. Storage Analysis (SC)*, Vol. 37. ACM, USA, 1–18.

[10] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Í nigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *ACM/IEEE Inter. Symp. Computer Arch. (ISCA)*, Vol. 51. ACM, USA, 118–132.

[11] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently Scaling Transformer Inference. In *Mach. Learning Sys. (MLSys)*, Vol. 5. ACM, USA, 606–624.

[12] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Rè, Ion Stoica, and Ce Zhange. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *Inter. Conf. Machine Learn. (ICML)* (Honolulu, HI), Vol. 202. PMLR, USA, 31094–31116.

[13] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[14] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2025. The Rise and Potential of Large Language Model Based Agents: A Survey. *Sci. China Info. Sci.* 68, 2 (Feb. 2025), 121101.

[15] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *USENIX Symp. Op. Sys. Des. Impl. (OSDI)*, Vol. 16. ACM, USA, 521–538.

[16] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2024. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL]

[17] Mingyu Zong and Bhaskar Krishnamachari. 2022. A survey on GPT-3. arXiv:2212.00857 [cs.CL]