# MagicDec: Breaking the Latency-Throughput Tradeoff for Long Context Generation with Speculative Decoding

Jian Chen *†, Vashisth Tiwari *†, Ranajoy Sadhukhan*†,
Zhuoming Chen†, Jinyuan Shi‡, Ian En-Hsu Yen‡, and Beidi Chen†♯

†Carnegie Mellon University
‡Moffett AI
♯Meta AI (FAIR)
{jianc2,vashistt,rsadhukh,zhuominc,beidic}@andrew.cmu.edu,
{jinyuan.shi,ian.yan}@moffett.ai

August 26, 2024

**Abstract**

Large Language Models (LLMs) have become more prevalent in long-context applications such as interactive chatbots, document analysis, and agent workflows, but it is challenging to serve long-context requests with low latency and high throughput. Speculative decoding (SD) is a widely used technique to reduce latency without sacrificing performance but the conventional wisdom suggests that its efficacy is limited to small batch sizes. In **MagicDec**, we show that surprisingly SD can achieve speedup even for a high throughput inference regime for *moderate to long sequences*. More interestingly, an intelligent drafting strategy can achieve *better speedup with increasing batch size* based on our rigorous analysis. MagicDec first identifies the bottleneck shifts with increasing batch size and sequence length, and uses these insights to deploy speculative decoding more effectively for high throughput inference. Then, it leverages draft models with sparse KV cache to address the KV bottleneck that scales with both sequence length and batch size. This finding underscores the broad applicability of speculative decoding in long-context serving, as it can *enhance throughput and reduce latency without compromising accuracy.* For moderate to long sequences, we demonstrate up to **2x** speedup for `LLaMA-2-7B-32K` and 1.84x speedup for `LLaMA-3.1-8 B` when serving batch sizes ranging from 32 to 256 on 8 NVIDIA A100 GPUs. The code is available at `https://github.com/Infini-AI-Lab/MagicDec/`.

## 1 Introduction

The landscape of language models has changed dramatically, with models capable of processing extremely long context lengths becoming increasingly prevalent [3, 6, 10, 16]. Interactive use cases such as chatbots [1] demand low latency, whereas background data-processing workloads prioritize high throughput [8, 20]. However, simultaneously achieving high throughput and low latency is challenging [2].

Speculative decoding (SD)[7, 15, 31] has emerged as a latency improvement technique which is guaranteed to maintain the generation quality. A fast draft model is used to generate multiple tokens and then the LLM verifies the speculated tokens in parallel, which is only marginally more expensive than decoding one token. However, SD has been believed to be less effective in improving throughput with larger batch sizes [17, 19, 26, 27]. On the other hand, methods like vLLM [14] and ORCA [34] can achieve high throughput by serving more requests, but cannot reduce latency. Lossy methods like quantization [11, 13, 32] and pruning [18, 28] have been proposed to improve both throughput and latency, but they can suffer from performance degradation. Given these trade-offs, we pose the following question:

---

*Equal contribution.

1
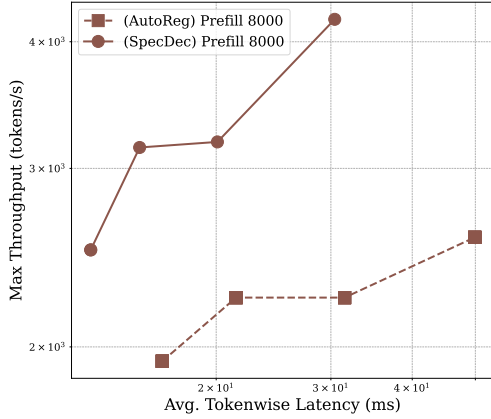
(a) *Throughput of Autoregressive decoding and SD for prefill 8000.*
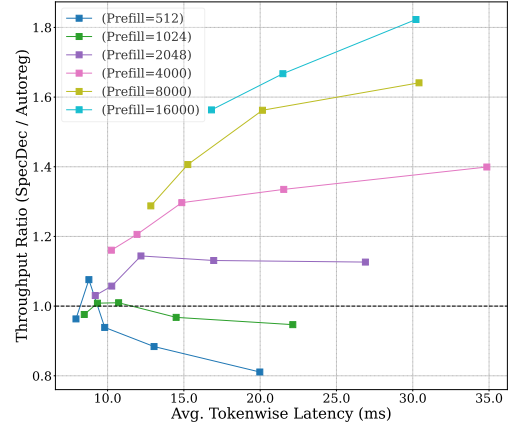


(b) *Throughput ratio of SD vs Autoregressive for various latency budgets.*

Figure 1: Throughput vs. Latency for `TinyLLama-1.1B` speculating `LLaMA-2-7B-32K` at different prefill lengths. **(a)** Throughput of autoregressive and SD against per-token latency for prefill 8000. **(b)** Throughput ratio of SD to autoregressive across latency budgets, showing that SD improves throughput for sequences longer than 1024.

> *Can we simultaneously* improve **throughput** *and* **latency** *without sacrificing* **accuracy**, *particularly for* long sequences?

In this work, we challenge the conventional belief that speculative decoding is inefficient for increasing throughput. We demonstrate that for moderate to large sequence lengths, ***speculative decoding can achieve all three objectives***: *increased throughput*, *reduced latency*, *and* lossless accuracy. Furthermore, by addressing the KV bottleneck using draft models with sparse KV cache, we can achieve even **better speedup for larger batches** of requests.

This conclusion is derived from a comprehensive analysis of LLM decoding performance, which reveals that as batch size grows, LLM decoding remains memory-bound for medium-to-long sequences[5, 27]. In this regime, KV cache becomes the dominant bottleneck. Unlike model parameter loading, this new bottleneck scales with batch size, making speculative decoding even more effective for large batches. As Figure 1 illustrates, for a given latency budget, SD has higher throughput than autoregressive decoding, and the efficacy of SD is evidently greater for longer sequences. Moreover, if we can allow a higher latency budget, we can improve throughput even more with MagicDec.

*[handwritten margin note: Due to KV cache bottleneck, LLM decoding is memory-bound at large seq. lengths as well]*

The main contributions of our work are:

1. We performed a theoretical analysis of LLM inference performance with speculative decoding. Our findings indicate that with medium-to-long sequence lengths and large batch sizes, LLM remains memory-bound and can be effectively accelerated through speculative decoding.

2. We found KV cache size of draft models, rather than model weights, is the most important factor in large batch and long sequence regime from our analysis. Therefore, we propose draft models with constant KV cache by StreamingLLM [33], making *improving speedup with increasing batch size* possible.

*[handwritten margin note: KV cache size of draft model is more important in large batch & long seq. length.]*

3. To verify our analysis, we conducted empirical evaluations across various GPUs and LLMs by sweeping batch sizes and sequence lengths. Our SD based system improves both throughput and latency, achieving a 2x speedup over autoregressive decoding for `LLaMA-2-7B-32K` and a 1.84x speedup for `LLaMA-3.1-8B` on 8 A100 GPUs .

## 2   Related Work

Numerous efforts have been made to improve the latency and throughput of LLMs. While methods like Flash-decoding [9], Flash-decoding++[12], FasterTransformers[21] have performed system optimizations

to improve latency, Speculative Decoding[7, 15, 31] has emerged as a novel sampling algorithm to speed up inference. On the other hand, batching has been a natural technique to improve GPU utilization and throughput. To make batching more effective, continuous batching [14, 22, 34] and chunked prefill [2] techniques have proposed intelligent batch scheduling techniques. Although these techniques can address the problems arising from heterogeneous batches with unequal context and generation lengths, they cannot solve the memory-bound problem of autoregressive decoding. In our work, we have considered the orthogonal direction of homogeneous batches, and therefore the aforementioned methods are complementary to our observation.

In our work, we have used speculative decoding as a method to balance the throughput-latency trade-off. We have particularly focused on draft models with StreamingLLM KV cache. This approach is motivated by the recent finding in Triforce [27] that self-speculation with sparse KV is an effective draft choice when KV loading is the main bottleneck. While Triforce is designed for small batches of extremely long sequences, we have focused on large batches of moderate to long sequences in our work.

Although promising for single batch requests, Speculative Decoding poses new challenges when implemented with batch support. Because the number of accepted tokens in SD follows a truncated geometric distribution [15], the average accepted length can vary throughout the batch, leading to misalignment of the sequence lengths. Qian *et al.*[24] have optimized the attention kernels to take care of this unequal number of accepted tokens across the batch. However, as previously reported, the excess computation for SD could be restrictive in a large batch size regime [17, 19, 26, 27]. Hence, [17, 26] suggest a reduction in speculation length with increasing batch size.

These findings imply that SD is not as effective in improving throughput; however, we notice that these observations are limited to a very small sequence-length regime. In our work, we focus on the relatively longer sequences that have been ubiquitous in practice.

# 3 Theoretical Analysis

In this section, we first outline the theoretical model to estimate the speed-up using SD and discuss various factors that affect it. In addition, we conduct several theoretical analysis on how the effect of these factors varies in different sequence length and batch size regimes. This forms the basis of our empirical studies discussed in section §5.

As introduced in §1, SD uses the LLM (target model) to verify the tokens speculated by a small draft model. This parallel verification amortizes the cost of loading the model parameters. The final output tokens are sampled using rejection sampling, ensuring the same output as the target distribution [7, 15].

## 3.1 Mathematical Formulation of Speedup

Given a speculation length $\gamma$ for a sequence of length $S$ and batch size $B$, let $\mathbf{T_T(B, S, 1)}$ and $\mathbf{T_D(B, S, 1)}$ denote the times taken by the target and draft models to decode one token, respectively. The verification time, $\mathbf{T_V(B, S, \gamma)}$, is the time it takes the target model to verify the $\gamma$ tokens speculated by the draft model in a single forward pass. Given an acceptance rate $\alpha \in [0, 1]$ and speculation length $\gamma$, $\mathbf{\Omega(\gamma, \alpha)}$ represents the expected number of tokens generated in one verification step, as described in [15].

$$\Omega(\gamma, \alpha) := \mathbb{E}(\# \text{ generated tokens}) = \frac{1 - \alpha^{\gamma+1}}{1 - \alpha} \tag{1}$$

The total time taken for speculative decoding, $T_{Total}^{SD}$, is given by:

$$T_{Total}^{SD} = \gamma \cdot T_D(B, S, 1) + T_V(B, S, \gamma)$$

The per-token latency for speculative decoding is simply $T_{Avg}^{SD} = \frac{T_{Total}^{SD}}{\Omega(\gamma, \alpha)}$.

For simplicity, we will refer to these times as $T_T$, $T_D$, and $T_V$ going forward, with the dependence on $B$ and $S$ implied unless otherwise specified. The total speedup from speculative decoding is then expressed as:

$$\frac{T_T}{T_{Avg}^{SD}} = \Omega(\gamma, \alpha) \cdot \frac{T_T}{\gamma \cdot T_D + T_V(\gamma)} \tag{2}$$
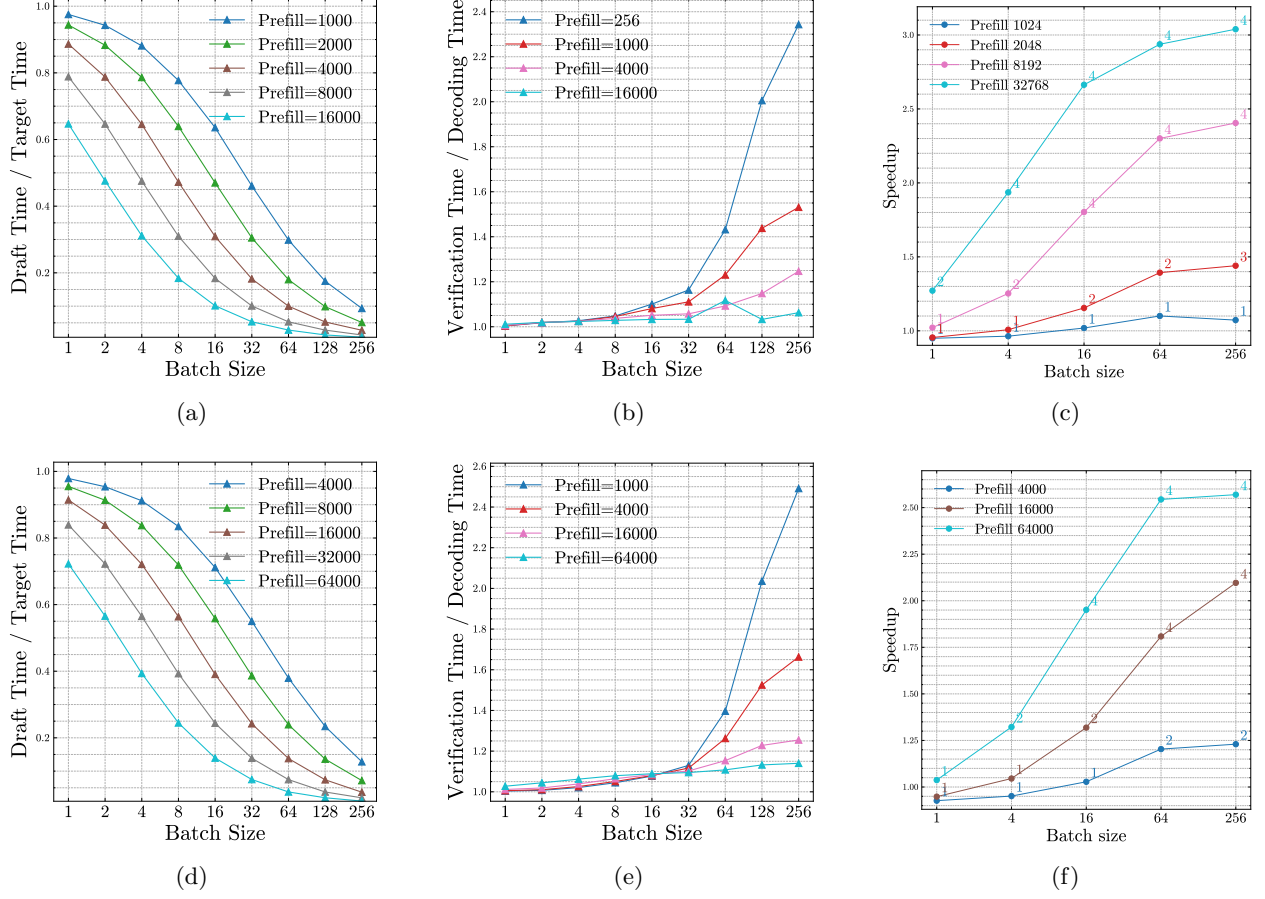
3

Figure 2: Theoretical analysis and expected speedup for `LLaMA-2-7B-32K` (top row) and `LLaMA-3.1-8B` (bottom row) models deployed on 8×A100s with $\gamma = 3$. **(a, d)** Theoretical $\frac{\mathbf{T_D}}{\mathbf{T_T}}$ versus batch sizes, **(b, e)** Theoretical $\frac{\mathbf{T_V}(\gamma)}{\mathbf{T_T}}$ versus batch size, and **(c, f)** Theoretical expected speedup with self-speculation for a budget of 512 across different batch sizes.

This equation can be further rewritten as:

$$\frac{T_{Avg}^{SD}}{T_T} = \frac{1}{\Omega(\gamma, \alpha)} \left( \frac{\gamma \cdot T_D}{T_T} + \frac{T_V(\gamma)}{T_T} \right) \tag{3}$$

## 3.2 Factors Affecting Speculative Decoding Speedup

Using Eq 3, we can see that speed-up depends on three primary factors: (a) draft to target cost ratio $\frac{T_D}{T_T}$ (b) verification to target decoding cost ratio $\frac{T_V(\gamma)}{T_T}$ and (c) expected generation length $\Omega(\gamma, \alpha)$.

***1. Draft to Target Cost Ratio*** $(T_D/T_T)$ **:**   A lightweight draft model is generally preferred for a shorter model loading time and fewer computations, leading to $T_D/T_T \approx 0$. However, with increasing sequence length, the bottleneck is shifted to KV cache loading. Furthermore, this bottleneck scales linearly with batch size, making KV loading an even bigger bottleneck for large batches as shown in Figure 3a. In the cases of `LLaMA-3.1-8B` for `LLaMA-3.1-70B` and `LLaMA-2-7B` for `LLaMA-2-70B`, the draft models can occupy up to $38 \sim 140\%$ memory footprint of target models (Figures 3b and 3c)due to the fact that $\dim_{kv}/\dim_{model}$ is higher. Hence, for long sequences, draft models with sparse KV is quite beneficial[27]. This can be seen in Figure 4a, which illustrates how $T_D/T_T$ for self-speculation approaches 0 with increasing sequence length for batch size 256.
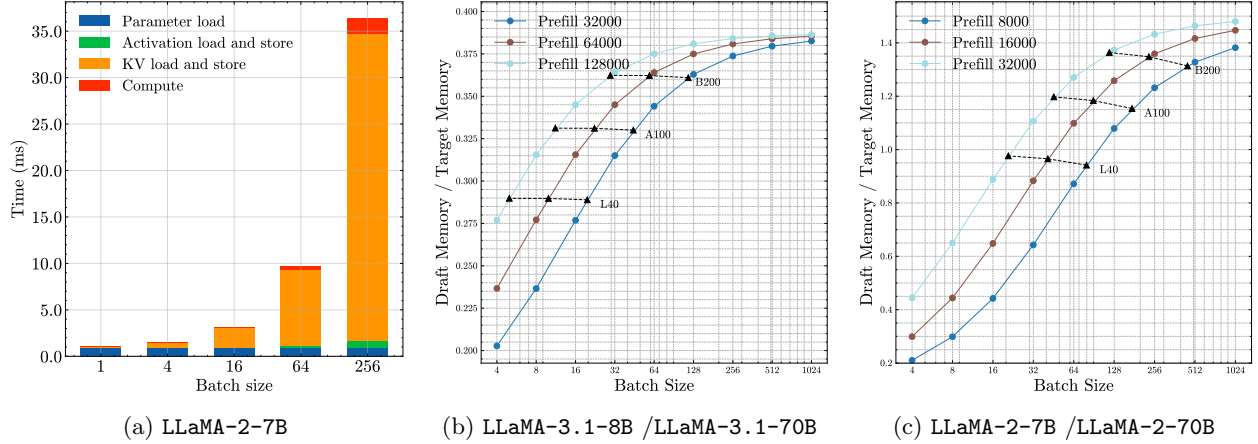
4

(a) `LLaMA-2-7B`      (b) `LLaMA-3.1-8B` /`LLaMA-3.1-70B`      (c) `LLaMA-2-7B` /`LLaMA-2-70B`

Figure 3: Theoretical analysis of KV cache bottleneck in speculative decoding. **(a)** Time breakdown of `LLaMA-2-7B` vs batch size (4096 prefill): KV cache, weights, activations loading, and compute. **(b,c)** Draft/target memory ratio vs batch size across different sequence lengths for `LLaMA-3.1-8B` /`LLaMA-3.1-70B` and `LLaMA-2-7B` /`LLaMA-2-70B` architectures.

In addition, Figure 2a and Figure 2d show how this ratio diminishes with increasing batch size for sufficiently long sequences. While the draft cost increases with larger batch sizes mainly due to increased computation time, the target cost rises even more due to the greater KV loading time. This makes the draft-to-target cost ratio decrease with increasing batch size, making speculative decoding more effective.

**2. Verification to Target Decoding Cost Ratio** $(T_V(\gamma)/T_T)$ **:** As evident from Eq 3, this ratio is desired to be close to 1. In low batch size setting, the verification process makes use of the underutilized compute and amortizes the cost of model loading. However, previous work [17] observed that with large batch sizes, $T_V(\gamma)/T_T$ becomes significantly greater than 1 as the model becomes more compute-bound [17, 19].

Our theoretical analysis challenges this argument. Figure 2b and Figure 2e illustrate how the ratio changes for `LLaMA-2-7B-32K` and `LLaMA-3.1-8B` with increasing batch size for different sequence lengths and a fixed speculation length of 3. As can be seen from the two figures, the $(T_V(\gamma)/T_T)$ ratio is expected to remain reasonably close to 1 for long sequences even when the batch size is quite large.

**3. The expected generation length** $(\Omega(\gamma, \alpha))$**:** $\Omega$ is a function of the acceptance rate $\alpha$ and the speculation length $\gamma$. Here we look at how $\alpha$ and $\gamma$ affect $\Omega$ and hence the speedup.

(a) $\underline{\alpha:}$ For a fixed draft cost, as $\alpha$ increases, $\Omega(\gamma, \alpha)$ increases, leading to a higher speed-up. In the regimes where $T_V(\gamma) \gg T_T$, $\alpha$ becomes pivotal as a lower $\alpha$ leads to considerable time spent on verifying tokens that are eventually discarded [17]. On the other hand, achieving a higher $\alpha$ usually comes at the cost of a higher $T_D$.

(b) $\underline{\gamma:}$ A longer speculation length increases $\Omega(\gamma, \alpha)$, but it also raises the verification and draft decode costs. In addition, the proportion of accepted tokens reduces with increasing $\gamma$ [17]. This is because $\Omega$ follows a truncated geometric distribution [15], as seen in eq. 1. Thus, finding the optimal $\gamma$ is crucial for achieving the best speedup under different batch sizes and sequence lengths.

## 3.3 Speculative Decoding Speedup Analysis

In this section, we examine the impact of increasing batch size on speculative decoding performance and discuss how our theoretical analysis provides insights into overcoming this challenge.

*Our analysis identifies a critical sequence length $\mathbf{S_{inflection}}$ for a given model pair and hardware. When $S \geq S_{inflection}$, the speculative decoding speedup tends to increase with batch size. However, this is not the case for $S < S_{inflection}$.*
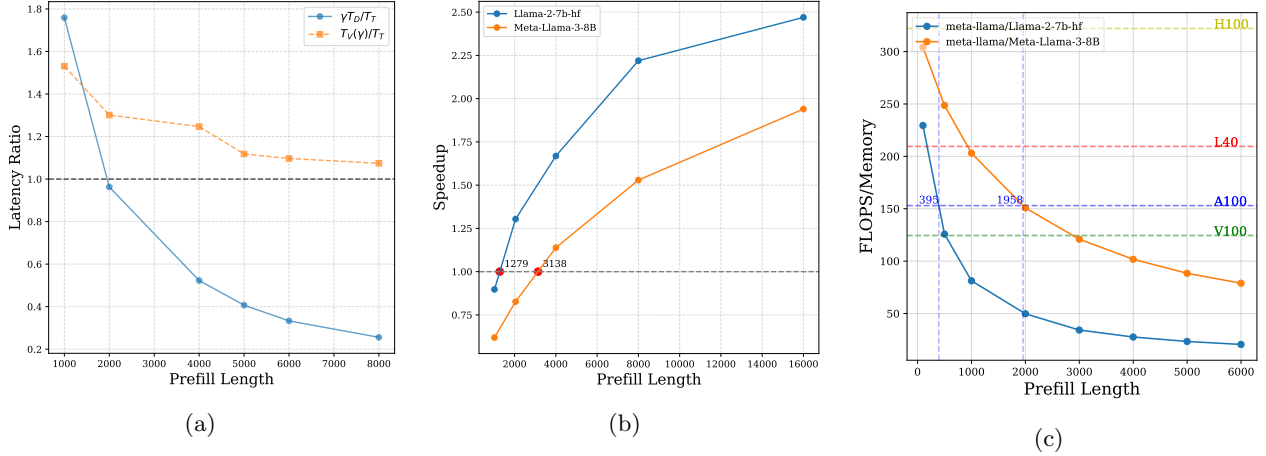
5

Figure 4: Theoretical analysis of self-speculation for `LLaMA-2-7B` and `LLaMA-3.1-8B` with a draft StreamingLLM budget of 512 and a batch size of 256. **(a)** Ratio of target-draft latency $\left(\frac{\gamma \cdot T_D}{T_T}\right)$ and verification-target latency $\left(\frac{T_V(\gamma)}{T_T}\right)$ versus sequence length for `LLaMA-2-7B-32K`, with $\gamma = 3$. **(b)** Theoretical speedup for different sequence lengths with a fixed $\alpha = 0.8$. **(c)** Theoretical arithmetic intensity for different sequence lengths and different models.

$S < S_{\text{inflection}}$: As noted in Subsection §3.2, a shorter verification time allows SD to achieve a greater speedup. While serving smaller batches, SD can efficiently use underutilized compute resources for its verification phase. However, large batches can saturate the available compute, making verification more expensive. If the draft token acceptance rate is low, the target model spends considerable time verifying incorrect speculations, reducing SD efficiency.

In this regime, our theoretical estimate aligns with Liu *et al.*[17]. As illustrated in Figure 2c, the speedup with SD decreases with batch size for prefill lengths below the critical sequence length.

$S \geq S_{\text{inflection}}$ : In this regime, we note that both $\frac{T_D}{T_T}$ and $\frac{T_V(\gamma)}{T_T}$ are low, favoring speculative decoding as per equation 3. For sufficiently long sequences, the ratio $\frac{T_D}{T_T}$ decreases with increasing batch size, as discussed in Section §3.2. Furthermore, the rate of increase in $\frac{T_V(\gamma)}{T_T}$ with the batch size is significantly slowed for longer sequences, while the acceptance rate remains unaffected. As a combined effect, there is an increase in speedup with batch size.

The key observation behind our analysis is that, for large sequences, KV becomes the main bottleneck instead of compute [5, 27]. This shift in bottleneck from compute to KV memory forces the target model to be memory bound even for a large batch size. Figure 4c shows for batch size 256 how the bottleneck changes from compute to memory with increasing sequence length (for different models and hardware pairs). As a result of this shift, the estimated ratios of verification and decoding time $\left(\frac{T_V(\gamma)}{T_T}\right)$ remain consistently close to 1 for $S > S_{\text{inflection}}$ (see Figure 2b). In addition, Figure 3a shows how KV loading becomes even more dominant with increasing batch size. If we use sparse KV cache for the draft model, the draft to target cost ratio will go down with increasing batch size. Consequently, we expect a greater speedup for larger batches. Thus, as Figure 2c illustrates in the case of `LLaMA-2-7B-32K`, the speedup with SD is expected to improve with increasing batch size for longer sequence lengths. It can be seen that the theoretical speedup decreases with batch size for $S < 1024$, but for $S \geq 1024$, the speedup increases with batch size.

**Factors Affecting $S_{\text{inflection}}$:** The critical sequence length, $S_{\text{inflection}}$, depends on both the model and the hardware. Figure 4c shows the FLOPS-to-memory ratio for `LLaMA-2-7B` and `LLaMA-3.1-8B`. For a device with a higher FLOPS-to-memory bandwidth ratio, we expect a lower $S_{\text{inflection}}$, as illustrated in Figure 4c. Additionally, the model itself influences $S_{\text{inflection}}$. For instance, `LLaMA-3.1-8B` has a higher $S_{\text{inflection}}$ due to its Grouped Query Attention (GQA), which requires a larger sequence length to achieve the same KV memory footprint (see Figure 4b).

6

# 4 Draft Model Design

As discussed in §3, with the increasing batch size and the growing sequence length, the KV cache becomes the bottleneck. Accurate draft models with constant KV cache are ideal. Similar to [27], we use StreamingLLM [33] for draft models due to its simplicity and effectiveness. It employs *attention sinks* (KV of initial tokens) with a sliding window KV for attention computation, enabling a stable, training-free method to handle infinitely long contexts. Specifically,

**Self-Speculation.** We leverage target models with StreamingLLM cache for accurate drafting since KV cache, rather than model weights, becomes major bottleneck. As we see in our theoretical analysis in Figure 4a for batch size 256, the relative cost of the self-speculation draft (with fixed budget 512) approaches 0 for large sequence lengths.

**Standalone GQA Draft Models.** Since the KV size is several times smaller than that of regular (MHA) models, GQA draft models can further reduce draft cost on the basis of StreamingLLM cache, when the batch size and sequence length are large.

# 5 Experiments

In this section, we demonstrate how speculative decoding increases speedup with batch size for sufficiently large sequence lengths, which matches our theoretical analysis. We conducted most experiments on 8 Nvidia A100 GPUs with 8-way Tensor Parallelism. We also tested on higher-end GPUs like Nvidia H100 and lower-cost alternatives like Nvidia L40, these results are shown in Table 2a and Table 2b. Across **all** these devices, we show that speculative decoding achieves significantly better throughput and lower latency than autoregressive decoding with long sequence lengths.

---

**Algorithm 1** GQA Custom

---

**Require:** Matrices for incoming query $Q \in \mathbb{R}^{H_q \times T \times d}$, incoming key and values $K, V \in \mathbb{R}^{H_k \times T \times d}$, past KV $K_{cache}, V_{cache} \in \mathbb{R}^{H_k \times L \times d}$
**Ensure:** Output matrix $Y \in \mathbb{R}^{H_q \times T \times d}$, updated $K_{cache}, V_{cache}$
  1: Define $G \leftarrow H_q/H_k$
  2: Flatten query heads $Q_{i \cdot G}, \ldots Q_{i \cdot G + G - 1}$ along sequence length dimension $\forall i \in \{0, \ldots, H_k - 1\}$ such that, $Q \in \mathbb{R}^{H_k \times (T \cdot G) \times d}$.
  3: Initialize $k_{new} = (0)_{H_k \times (T \cdot G) \times d}$, $v_{new} = (0)_{H_k \times (T \cdot G) \times d}$, $offset \leftarrow (0)_{T \cdot G}$
  4: $k_{new}[:, i \cdot G, :] \leftarrow k[:, i, :], v_{new}[:, i \cdot G, :] \leftarrow v[:, i, :] \quad \forall i \in \{0, \ldots, T - 1\}$ {#zero vectors interleaved to simulate block-diagonal masking}
  5: $offset[i \cdot G] \leftarrow i \cdot (G - 1) \quad \forall i \in \{0, \ldots, T - 1\}$ {#zero key vectors add bias to partition function}
  6: $y, lse \leftarrow \texttt{flash\_attn\_with\_kvcache}(q, k_{cache}, v_{cache}, k_{new}, v_{new})$
  7: $correction \leftarrow 1/(1 - offset \cdot \exp(-lse))$
  8: $y \leftarrow y \cdot correction$
  9: Reshape $y$ back to $(H_q, T, d)$
 10: Update $k_{cache}, v_{cache}$ with new $k, v$
 11: **return** $y$

---

## 5.1 Experimental Setup

In our experiments, we focused on two types of draft models as we discussed in Section 4. We tested various StreamingLLM cache budgets to balance draft cost and acceptance rate. These experiments were performed on the PG-19 dataset [25]. Each run was evaluated on **50 samples**, generating **64** tokens per sentence in the batch using greedy decoding. The specifics of our draft selection are as follows:

  1. **Self-Speculation:** We experimented with `LLaMA-2-7B-32K` [29, 30] and `LLaMA-3.1-8B-128K` [3] models with various StreamingLLM budgets for drafting.

2. **Standalone GQA Model:** We used the long context variant of `LLaMA-2-7B`, `LLaMA-2-7B-32K` [29], as the target model and `TinyLLaMA-1.1B` [35] for drafting.

## 5.2   System Implementation

We built our speculative decoding system on top of GPT-Fast [23]. FlashAttention-2 [9] was used to accelerate attention computation. During implementation, we observed that Grouped-Query Attention (GQA) [4] lacks adequate support from FlashAttention when verifying multiple tokens simultaneously. To address this, we modified the GQA implementation to achieve the desired speed improvements detailed below (see Algorithm 1). Additionally, we used Pytorch CUDA graphs to reduce CPU-side kernel launching overhead. For self-speculation and draft models with limited context length, we used StreamingLLM [33] with fixed KV cache budgets. This predetermined cache budget helps save CPU overhead through CUDA graphs.

**Optimized Group Query Attention:** The implementation of Group Query Attention (GQA) by FlashAttention[9] is found to be poorly optimized for the verification of multiple tokens. However, there is no such problem with regular Multi-Head Attention (MHA). Thus, to circumvent the problem of slow verification, we utilized FlashAttention's MHA to implement GQA. The implementation details can be found in Algorithm 1.
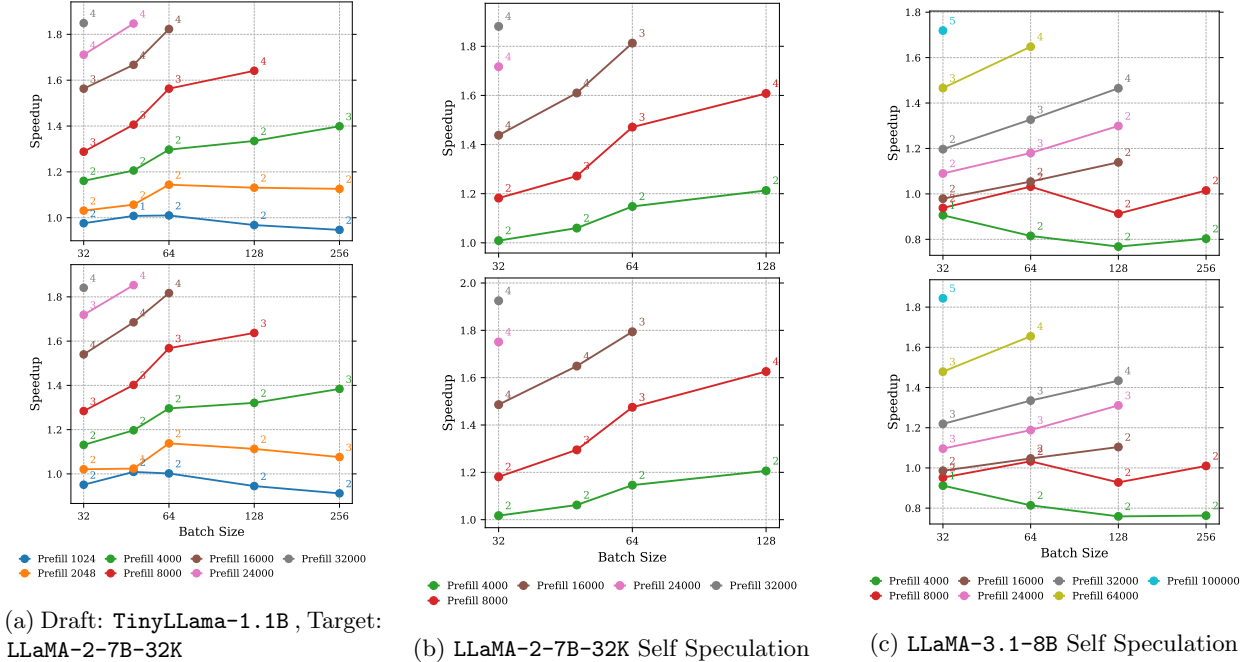


(a) Draft: `TinyLLama-1.1B`, Target: `LLaMA-2-7B-32K`

(b) `LLaMA-2-7B-32K` Self Speculation

(c) `LLaMA-3.1-8B` Self Speculation

Figure 5: End-to-end speedups for target-draft pairs across various StreamingLLM budgets (top: 256, bottom: 512). Annotations indicate $\gamma_{\text{optimal}}$, the $\gamma$ value corresponding to the highest speedup achieved, with $\gamma \in \{1, 2, 3, 4\}$ for `LLaMA-2-7B-32K` and $\gamma \in \{1, 2, 3, 4, 5\}$ for `LLaMA-3.1-8B` targets.

## 5.3   Results

The main results of the experiment are summarized in Figures 5. These figures show the speedup achieved by speculative decoding at the optimal speculation length ($\gamma_{\text{optimal}}$) across various batch sizes and sequence lengths.

We can find that speculative decoding consistently outperforms autoregressive decoding except when the batch size is large and the sequence length is short. Moreover, as the sequence length increases, the speedup grows with batch size, achieving both higher throughput and lower latency. With optimal speculation lengths,

#### (a) LLaMA-2-7B-32K , TinyLLama-1.1B

| S | B | $\gamma T_D$ | $T_V$ | $\Omega$ | $T^{AR}$ | $T^{SD}$ | x |
|---|---|---|---|---|---|---|---|
| 1024 | 32 | 8.21 | 9.55 | 2.19 | 8.27 | 8.70 | 0.95 |
| 1024 | 48 | 8.46 | 10.66 | 2.19 | 9.41 | 9.33 | 1.01 |
| 1024 | 64 | 9.26 | 13.05 | 2.19 | 10.83 | 10.80 | 1.00 |
| 1024 | 128 | 12.04 | 18.87 | 2.19 | 14.02 | 14.83 | 0.94 |
| 4000 | 32 | 8.46 | 13.21 | 2.19 | 11.89 | 10.52 | 1.13 |
| 4000 | 48 | 8.71 | 16.19 | 2.19 | 14.39 | 12.02 | 1.20 |
| 4000 | 64 | 9.35 | 21.83 | 2.19 | 19.28 | 14.88 | 1.30 |
| 4000 | 128 | 12.31 | 33.82 | 2.19 | 28.77 | 21.78 | 1.32 |
| 8000 | 32 | 8.61 | 18.40 | 2.18 | 16.53 | 13.02 | 1.27 |
| 8000 | 48 | 8.91 | 23.67 | 2.18 | 21.45 | 15.58 | 1.38 |
| 8000 | 64 | 9.58 | 34.32 | 2.18 | 31.49 | 20.80 | 1.51 |
| 8000 | 128 | 12.54 | 53.78 | 2.18 | 49.89 | 31.25 | 1.60 |
| 16000 | 32 | 8.78 | 27.79 | 2.17 | 26.28 | 17.46 | 1.50 |
| 16000 | 48 | 9.33 | 38.29 | 2.18 | 35.83 | 22.52 | 1.59 |
| 16000 | 64 | 9.92 | 58.14 | 2.17 | 55.08 | 31.99 | 1.72 |
| 24000 | 32 | 8.68 | 37.57 | 2.16 | 35.70 | 22.05 | 1.62 |
| 24000 | 48 | 9.31 | 52.89 | 2.16 | 50.28 | 29.48 | 1.71 |
| 32000 | 32 | 8.83 | 47.35 | 2.17 | 44.94 | 26.55 | 1.69 |

#### (b) LLaMA-2-7B-32K Self Speculation

| S | B | $\gamma T_D$ | $T_V$ | $\Omega$ | $T^{AR}$ | $T^{SD}$ | x |
|---|---|---|---|---|---|---|---|
| 4000 | 32 | 15.42 | 13.17 | 2.56 | 11.89 | 11.69 | 1.02 |
| 4000 | 48 | 16.96 | 16.38 | 2.56 | 14.39 | 13.55 | 1.06 |
| 4000 | 64 | 19.75 | 22.01 | 2.57 | 19.28 | 16.82 | 1.15 |
| 4000 | 128 | 25.82 | 33.79 | 2.56 | 28.77 | 23.86 | 1.21 |
| 8000 | 32 | 15.70 | 18.23 | 2.53 | 16.53 | 13.99 | 1.18 |
| 8000 | 48 | 18.44 | 24.32 | 2.53 | 21.45 | 17.50 | 1.23 |
| 8000 | 64 | 20.03 | 34.30 | 2.53 | 31.49 | 22.05 | 1.43 |
| 8000 | 128 | 26.10 | 53.69 | 2.52 | 49.89 | 32.25 | 1.55 |
| 16000 | 32 | 16.06 | 27.54 | 2.50 | 26.28 | 18.02 | 1.46 |
| 16000 | 48 | 19.75 | 39.03 | 2.50 | 35.83 | 24.15 | 1.48 |
| 16000 | 64 | 20.87 | 58.15 | 2.51 | 55.08 | 32.16 | 1.71 |
| 24000 | 32 | 15.80 | 37.06 | 2.49 | 35.70 | 21.77 | 1.64 |
| 32000 | 32 | 16.19 | 46.55 | 2.50 | 44.94 | 25.64 | 1.75 |

#### (c) LLaMA-3.1-8B Self Speculation

| S | B | $\gamma T_D$ | $T_V$ | $\Omega$ | $T^{AR}$ | $T^{SD}$ | x |
|---|---|---|---|---|---|---|---|
| 4000 | 32 | 13.16 | 10.32 | 2.54 | 8.83 | 9.78 | 0.90 |
| 4000 | 64 | 16.48 | 13.55 | 2.54 | 10.07 | 12.36 | 0.81 |
| 4000 | 128 | 23.41 | 19.77 | 2.54 | 13.42 | 17.70 | 0.76 |
| 4000 | 256 | 39.29 | 35.05 | 2.53 | 23.23 | 30.46 | 0.76 |
| 8000 | 32 | 13.28 | 11.34 | 2.50 | 9.90 | 10.40 | 0.95 |
| 8000 | 64 | 16.98 | 16.06 | 2.51 | 14.16 | 13.72 | 1.03 |
| 8000 | 128 | 23.59 | 24.84 | 2.51 | 18.53 | 19.97 | 0.93 |
| 8000 | 256 | 39.32 | 46.44 | 2.51 | 35.35 | 34.99 | 1.01 |
| 16000 | 32 | 14.46 | 14.00 | 2.47 | 11.93 | 12.10 | 0.99 |
| 16000 | 64 | 18.00 | 21.15 | 2.48 | 17.17 | 16.40 | 1.05 |
| 16000 | 128 | 25.77 | 34.82 | 2.46 | 28.00 | 25.36 | 1.10 |
| 32000 | 32 | 14.12 | 19.04 | 2.46 | 17.13 | 14.05 | 1.22 |
| 32000 | 64 | 19.08 | 30.86 | 2.45 | 26.99 | 21.03 | 1.28 |
| 32000 | 128 | 28.26 | 54.98 | 2.45 | 47.24 | 34.94 | 1.35 |
| 64000 | 32 | 14.92 | 28.88 | 2.40 | 26.96 | 18.91 | 1.43 |
| 64000 | 64 | 18.25 | 50.19 | 2.40 | 46.09 | 29.22 | 1.58 |
| 100000 | 32 | 15.10 | 39.84 | 2.45 | 37.70 | 23.05 | 1.64 |

Table 1: Comparison of results for different LLaMA models and configurations (budget=512 and $\gamma = 2, 8\times$ A100). Here S and B represent prefill length and batch size, respectively.

#### (a) Results on 8 × L40, StreamingLLM budget for the draft model is 512, each with the optimal $\gamma$

| Target | Draft | Prefill | Bsz | $\gamma$ | $\gamma T_D(1)$ | $T_V(\gamma)$ | $\Omega(\gamma, \alpha)$ | $T^{AR}$ | $T^{SD}$ | x |
|---|---|---|---|---|---|---|---|---|---|---|
| Llama3.1-8B | Selfspec | 32000 | 32 | 3 | 44.11 | 45.12 | 3.00 | 36.62 | 30.32 | 1.21 |
| Llama2-7B | Selfspec | 8000 | 32 | 2 | 29.06 | 42.02 | 2.53 | 35.13 | 28.70 | 1.22 |
| Llama2-7B | Tinyllama1.1B | 8000 | 32 | 3 | 12.01 | 42.96 | 2.53 | 22.32 | 56.48 | 1.57 |
| Llama2-7B | Selfspec | 8000 | 64 | 3 | 58.33 | 74.85 | 3.14 | 62.92 | 42.96 | 1.46 |
| Llama2-7B | Tinyllama1.1B | 8000 | 64 | 3 | 19.66 | 74.91 | 2.51 | 62.92 | 38.33 | 1.64 |

#### (b) Results on 4 × H100, StreamingLLM budget for the draft model is 256, each with the optimal $\gamma$

| Target | Draft | Prefill | Bsz | $\gamma$ | $\gamma T_D(1)$ | $T_V(\gamma)$ | $\Omega(\gamma, \alpha)$ | $T^{AR}$ | $T^{SD}$ | x |
|---|---|---|---|---|---|---|---|---|---|---|
| Llama3.1-8B | Selfspec | 32000 | 16 | 2 | 9.42 | 12.46 | 2.36 | 11.90 | 9.60 | 1.24 |
| Llama3.1-8B | Selfspec | 32000 | 32 | 3 | 15.09 | 18.30 | 2.82 | 17.32 | 12.16 | 1.42 |
| Llama2-7B | Selfspec | 8000 | 32 | 3 | 14.20 | 15.64 | 2.98 | 14.85 | 10.29 | 1.44 |
| Llama2-7B | Tinyllama1.1B | 8000 | 32 | 3 | 6.57 | 15.65 | 2.48 | 14.85 | 9.28 | 1.60 |
| Llama2-7B | Selfspec | 8000 | 64 | 4 | 23.63 | 27.90 | 3.37 | 26.17 | 15.58 | 1.68 |
| Llama2-7B | Tinyllama1.1B | 8000 | 64 | 4 | 9.51 | 27.87 | 2.69 | 26.17 | 14.25 | 1.84 |

Table 2: Results on L40 and H100

SD achieves a **2**x speedup over autoregressive decoding for LLaMA-2-7B-32K self-speculation at a sequence length of 32k and batch size 32, and a **1.84**x speedup for LLaMA-3.1-8B at a sequence length of 100k and batch size 32. These combinations of sequence length and batch size correspond to the maximum KV size that fits within our experimental setup as detailed in Section §5.1.

The raw values for the verification, draft, and target latencies for specific StreamingLLM budgets and $\gamma$ are reported to support our discussion (Table 1). The results on 8 L40 and 4 H100 GPUs are also shown in Tables 2a and 2b, demonstrating that speculative decoding performs well when the batch size and sequence length are large on different types of GPUs. These tables also illustrate the influence of FLOPS-to-memory bandwidth ratios of different devices on speedup.

For most cases in our experiment, the relative gain in acceptance rate for budget 1024 over budgets 256 and 512 is not a good tradeoff. However, for longer sequences, as the relative size of the target KV is the main bottleneck, a higher $\alpha$ of a higher budget can be an important factor in providing speedup, as $\Omega(\gamma, \alpha)$ is higher. Notably, our **2x** speedup comes for batch size 32 and sequence length 32k for the LLaMA-2-7B-32K self-speculation setting for the highest StreamingLLM budget size of 1024 (Figure 6b).

Based on these results, we infer the following:

1. As hypothesized in §3.3, this inflection is evident in Figure 5. SD does not provide speedup with increased batch size for LLaMA-2-7B-32K and LLaMA-3.1-8B when $S < 2000$ and $S < 8000$, respectively. However, for sequences longer than these inflection points, we observe consistent improvements in speedups with higher batch sizes. These empirical values align well with the expectations on Figure 4b.

The factors affecting $S_{\text{inflection}}$ are discussed in detail in §3. We present the raw numbers in Tables 1 for 8 A100 GPUs.

2. Conventionally, $\gamma_{\text{optimal}}$ is expected to decrease with batch size [17]. However, this is not universal. As shown in Figures 5b and 5a, for $S > S_{\text{inflection}}$, $\gamma_{\text{optimal}}$ can increase with larger batch sizes. Similarly, an improved speedup is observed with batch size increasing in this regime.

3. Based on Figure 5 for A100 and Table 2b for H100, we observe a higher speedup on the H100 device. This is because the H100 has a higher FLOPS-to-memory bandwidth ratio than the A100, leading to $T_V \approx T_T$. Additionally, the higher compute bandwidth of the H100 reduces $T_D$, resulting in better speedup.

*Since lot of compute (memory bound regime) hence, $T_V \cong T_T$*

# 6 Conclusion

This work reassesses the trade-off between throughput and latency in long-context scenarios. We show that *speculative decoding can enhance throughput, reduce latency, and maintain accuracy.* Our theoretical and empirical analysis reveals that as the sequence length and batch size increase, bottlenecks shift from being compute-bound to memory-bound. This shift enables effective use of speculative decoding for longer sequences, even with large batch sizes, achieving up to **2x** speedup for `LLaMA-2-7B-32K` and **1.84x** for `LLaMA-3.1-8B` on 8 A100 GPUs. More surprisingly, larger batches can achieve even better speedup from speculative decoding if draft models with sparse KV cache are used. These results highlight the need to integrate speculative decoding into throughput optimization systems as long-context workloads become more common.

# References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve, 2024. URL `https://arxiv.org/abs/2403.02310`.

[3] AI@Meta. The llama 3 herd of models, 2024. URL `https://ai.meta.com/research/publications/the-llama-3-herd-of-models`.

[4] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL `https://arxiv.org/abs/2305.13245`.

[5] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale, 2022. URL `https://arxiv.org/abs/2207.00032`.

[6] Anthropic. Introducing the next generation of claude, 2024. URL `https://www.anthropic.com/news/claude-3-family`.

[7] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

[8] Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. Spreadsheetcoder: Formula prediction from semi-structured context, 2021. URL `https://arxiv.org/abs/2106.15339`.

[9] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. URL https://arxiv.org/abs/2307.08691.

[10] Google Deepmind. Our next-generation model: Gemini 1.5, 2024. URL https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/#build-experiment.

[11] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

[12] Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Hanyu Dong, and Yu Wang. Flashdecoding++: Faster large language model inference on gpus. *arXiv preprint arXiv:2311.01282*, 2023.

[13] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization, 2024. URL https://arxiv.org/abs/2401.18079.

[14] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Yu, Joseph E Gonzalez, Hao Zhang, and Ion Stoica. vllm: Easy, fast, and cheap llm serving with pagedattention. *See https://vllm.ai/ (accessed )*, 2023.

[15] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. *arXiv preprint arXiv:2211.17192*, 2022.

[16] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023. URL https://arxiv.org/abs/2304.08485.

[17] Xiaoxuan Liu, Cade Daniel, Langxiang Hu, Woosuk Kwon, Zhuohan Li, Xiangxi Mo, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Optimizing speculative decoding for serving large language models using goodput, 2024. URL https://arxiv.org/abs/2406.14066.

[18] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models, 2023. URL https://arxiv.org/abs/2305.11627.

[19] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*, 2023.

[20] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. Can foundation models wrangle your data?, 2022. URL https://arxiv.org/abs/2205.09911.

[21] NVIDIA. Fastertransformer. URL https://github.com/NVIDIA/FasterTransformer.

[22] Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vattention: Dynamic memory management for serving llms without pagedattention, 2024. URL https://arxiv.org/abs/2405.04437.

[23] pytorch-labs. Gpt-fast, 2023. URL https://github.com/pytorch-labs/gpt-fast.

[24] Haifeng Qian, Sujan Kumar Gonugondla, Sungsoo Ha, Mingyue Shang, Sanjay Krishna Gouda, Ramesh Nallapati, Sudipta Sengupta, Xiaofei Ma, and Anoop Deoras. Bass: Batched attention-optimized speculative sampling, 2024. URL https://arxiv.org/abs/2404.15778.

[25] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling, 2019. URL https://arxiv.org/abs/1911.05507.

[26] Qidong Su, Christina Giannoula, and Gennady Pekhimenko. The synergy of speculative decoding and batching in serving large language models, 2023. URL https://arxiv.org/abs/2310.18813.

[27] Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding, 2024. URL https://arxiv.org/abs/2404.11912.

[28] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models, 2024. URL https://arxiv.org/abs/2306.11695.

[29] TogetherAI. Preparing for the era of 32k context: Early learnings and explorations, 2023. URL https://www.together.ai/blog/llama-2-7b-32k.

[30] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

[31] Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation, 2023. URL https://arxiv.org/abs/2203.16487.

[32] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models, 2024. URL https://arxiv.org/abs/2211.10438.

[33] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453.

[34] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 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, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/yu.

[35] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model, 2024.

# A   Appendix

## A.1   Additional Results
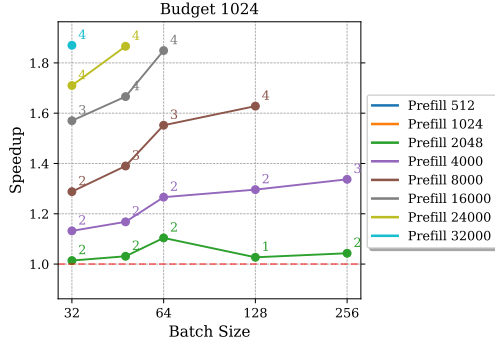


(a) `TinyLLama-1.1B`
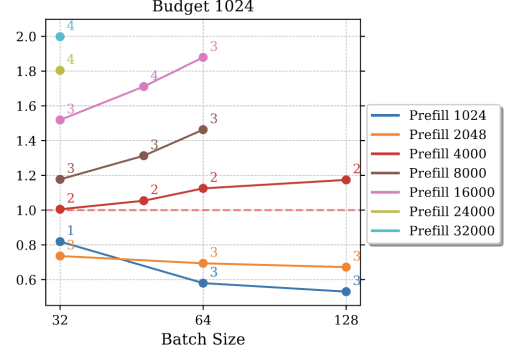
(b) `LLaMA-2-7B`

Figure 6: End-to-end Speedups for `LLaMA-2-7B-32K` with `TinyLLama-1.1B` and self-speculation draft with StreamingLLM of budget 1024. Annotations indicate $\gamma_{\text{optimal}}$ where $\gamma_{\text{optimal}} \in \{1, 2, 3, 4\}$