TETRIS: Optimal Draft Token Selection for Batch Speculative Decoding

Zhaoxuan Wu[*1], **Zijian Zhou**[*1,2], **Arun Verma**[1], **Alok Prakash**[1], **Daniela Rus**[1,3], **Bryan Kian Hsiang Low**[1,2]

[1]Singapore-MIT Alliance for Research and Technology, Republic of Singapore
[2]Dept. of Computer Science, National University of Singapore, Republic of Singapore
[3]CSAIL, Massachusetts Institute of Technology, USA

Correspondence: lowkh@comp.nus.edu.sg

Abstract

We propose TETRIS, a novel method that optimizes the *total throughput* of batch speculative decoding in multi-request settings. Unlike existing methods that optimize for a single request or a group of requests as a whole, TETRIS actively selects the most promising draft tokens (for every request in a batch) to be accepted when verified in parallel, resulting in fewer rejected tokens and hence less wasted computing resources. Such an effective resource utilization to achieve fast inference in large language models (LLMs) is especially important to service providers with limited inference capacity. Compared to baseline speculative decoding, TETRIS yields a consistently higher acceptance rate and more effective utilization of the limited inference capacity. We show theoretically and empirically that TETRIS outperforms baseline speculative decoding and existing methods that dynamically select draft tokens, leading to a more efficient batch inference in LLMs.

1 Introduction

Transformer-based large language models (LLMs) have shown remarkable abilities to solve different tasks across various domains, such as natural language (Zhao et al., 2023), computer vision (Yin et al., 2024), robotics (Zeng et al., 2023), code generation (Rozière et al., 2024), among others (Maslej et al., 2024). However, the autoregressive nature of LLMs (i.e., generating one token at a time) leads to an increasingly sluggish inference speed as the model size increases.

To address this problem, a recent widely-used approach is speculative decoding (SD) (Cai et al., 2024; Cheng et al., 2024; Leviathan et al., 2023; Li et al., 2024a,b): It achieves faster inference by using a *small draft model* to rapidly generate a sequence of *(draft) tokens* and then a *large target model* to verify whether to accept or reject them in parallel. When a token is rejected, the draft model generates a new sequence of tokens in the next step, starting from the most recently accepted token. A key aspect of SD is to determine the optimal number of draft tokens (i.e., *draft window size*) to generate and verify in each step. Generating more draft tokens allows the target model to verify a longer sequence at once (given sufficient computing resources/capacity for parallel inferences), which can potentially boost inference speed. However, doing so increases the risk of wasting computing resources since all tokens following the first rejected token must be discarded. In contrast, generating fewer draft tokens reduces this risk but limits the potential benefit of SD since the computing resources are not effectively utilized. Therefore, the optimal selection of draft tokens that would be accepted when verified by the target model in parallel is critical to improving both inference speed and resource utilization (Liu et al., 2024d).

Most existing works have focused on optimizing draft token selection for individual user requests (Agrawal et al., 2024; Huang et al., 2024; Liu et al., 2024c; Mamou et al., 2024), but may not work well for profit-driven LLM inference service providers who must manage multiple user requests under a limited inference capacity. Moreover, LLM inference service providers typically charge users based on the number of tokens served (Fireworks AI, 2025; Replicate, 2025). Hence, they are incentivized to maximize the total number of tokens served (i.e., *throughput*) across all user requests while ensuring fast response time to meet service level agreement (Wieder et al., 2011). So, they would employ computing clusters to process large batches of user requests simultaneously and use SD to further improve the inference speed.

Such batch processing of user requests entails a fundamentally different optimization objective for SD compared to handling individual requests. For SD of a single request, supposing a fast draft model

* Equal contribution.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 33329–33345

Figure 1: Standard SD (left) uses a fixed draft window size, while TETRIS (right) generates extra draft tokens and dynamically optimizes draft token selection for every request in a batch, resulting in more accepted tokens.

with negligible runtime, the objective is to maximize the draft window size as long as the target model can verify all draft tokens in parallel by fully utilizing the inference capacity. It can be naively extended to batch processing by widening the draft window for all requests until the inference capacity is reached. This is inefficient as each request may require a different optimal draft token selection due to varying difficulty in speculation (i.e., generating tokens to match the target model's outputs).

This paper presents a theoretical framework that dynamically optimizes the draft token selection for every user request from the perspective of a capacity-limited LLM inference service provider who aims to maximize resource utilization. Since draft token verification is the most time-consuming component of SD, we propose **TETRIS**, a method that greedily selects draft tokens with a high likelihood of acceptance by the target model. The name of our method is derived from the shape of its selected tokens, as shown in Fig. 1. We demonstrate that TETRIS strictly outperforms standard SD by achieving higher total throughput. Our work bridges a critical yet overlooked gap in current research, allowing service providers to improve total throughput with batch SD. The specific contributions of our work here are summarized below:

- In Sec. 3, we introduce the problem of optimal draft token selection in multi-request settings, and in Sec. 4.1, we propose TETRIS, a novel method that selects optimal draft tokens in log-linear time for the target model's verification.
- In Sec. 4.2, we theoretically show that TETRIS achieves optimal throughput at each decoding step and globally in the absence of drafting time (i.e., time to generate draft tokens) under reasonable token acceptance assumptions.
- In Sec. 5, our empirical results show that TETRIS consistently outperforms standard SD and existing methods that use dynamic draft windows for a batch in terms of total throughput and end-to-end latency (including drafting time), highlighting the potential of TETRIS to improve inference speed in real-world model service deployments.

2 Related Work

Speculative Decoding (SD). By employing a draft-then-verify strategy for lossless accelerations of LLM inference, SD has attracted significant attention recently (Ryu and Kim, 2024; Xia et al., 2024). Recent advancements based on SD have focused on developing more efficient draft models by producing multiple drafts for the next few tokens (Cai et al., 2024; Cheng et al., 2024; Li et al., 2024b). Additionally, some methods have optimized the speculation accuracy by aligning the draft model with the target model (Liu et al., 2024e; Zhou et al., 2024a) or leveraging the target model itself to draft via techniques like layer skipping (Zhang et al., 2024). To facilitate more efficient verification, tree attention has been proposed for speedy tree-structured candidate verification (Miao et al., 2024; Spector and Re, 2023). In contrast, our work explores a complementary approach that intervenes between the draft and target models, performing strategic draft token selection to improve throughput over batched requests. Our method can be seamlessly integrated with the above techniques for a more efficient SD system.

LLM Scheduling. With the growing popularity of LLM as a service, several works have considered improvements to the scheduling of LLM services. These works can be broadly categorized into client-side and server-side approaches. Server-side approaches (Fu et al., 2024; Kim et al., 2024; Liu et al., 2024d; Wang et al., 2024a) have focused on increasing the throughput of LLM services, which may lead to an unfair allocation of inference re-

sources to users, hence causing starvation. On the other hand, client-side approaches (Liu et al., 2024b; Sheng et al., 2024) have focused on improving user satisfaction by improving client-side metrics (e.g., decreasing maximal waiting time or end-to-end latency). Our work considers the scenario where the LLM inference service provider employs SD to ensure user satisfaction with inference speed while simultaneously aiming to maximize service throughput to optimize profitability.

Draft Window Optimization. In the foundational paper on SD, the authors have proposed to generate a window of draft tokens (Leviathan et al., 2023). The optimal draft window is theoretically determined under an impractical assumption of identical conditional acceptance rates for all draft tokens (Leviathan et al., 2023). Empirically, such an acceptance rate can be estimated by a moving average of past requests (Liu et al., 2024d). Other heuristics for finding the optimal draft window include stopping the draft generation when the draft model's confidence score falls below a predetermined threshold (Kim et al., 2023; Liu et al., 2024a) or when an entropy-controlled criterion is met (Agrawal et al., 2024). Cai et al. (2024) have proposed taking the union of these two heuristics. These existing works have operated at a single-request level, except that of Liu et al. (2024d) which adaptively determines a single draft window for all requests in a batch. Note that considering each request independently or using a common draft window for a batch can lead to inefficiencies in allocating verification budgets (i.e., inference capacity) across multiple requests, especially when operating under the limited computing resources of an LLM inference service provider.

3 Problem Setup

This section first introduces speculative decoding and then describes the optimal draft token selection problem and the performance metrics used.

3.1 Speculative Decoding (SD)

SD is an efficient inference method designed to accelerate the decoding process in LLMs and involves two phases: drafting followed by verification. Initially, a lightweight draft model, denoted as $\mathcal{S}$, quickly generates candidate draft tokens. Subsequently, these tokens are verified against the generations from the target model, denoted as $\mathcal{M}$, which is also often referred to as the verification model. SD allows parallelized verifications of tokens by $\mathcal{M}$, as opposed to the conventional autoregressive decoding used in language models. Hence, SD yields significant improvement in decoding speed.

Specifically, the draft model generates k draft tokens $d_1, \ldots, d_k$ in an autoregressive manner where k is the draft window size. Given a prompt or prefix x, the generation process follows $d_i \sim p_{\mathcal{S}}(\cdot|x, d_1, \ldots, d_{i-1})$. For notational simplicity, we denote $p_{\mathcal{S}}(d_i) = p_{\mathcal{S}}(d_i|x, d_1, \ldots, d_{i-1})$. The verification follows a rejection sampling procedure. If $p_{\mathcal{S}}(d_i) \leq p_{\mathcal{M}}(d_i)$, the draft token d_i is accepted. Otherwise, we reject the draft token with a probability of $1 - p_{\mathcal{M}}(d_i)/p_{\mathcal{S}}(d_i)$ and then output a new token sampled from an adjusted distribution $p_{\mathcal{M}}(d_i') = \mathrm{norm}(\max(0, p_{\mathcal{M}}(d_i') - p_{\mathcal{S}}(d_i')))$, where $\mathrm{norm}(\cdot)$ normalizes the probability distribution. Hence, the acceptance of draft tokens depends on both $p_{\mathcal{S}}(\cdot)$ and $p_{\mathcal{M}}(\cdot)$ and plays a vital role in the effectiveness of SD. A higher acceptance suggests the possibility of greater speedup gain with a larger k. We defer a more detailed discussion of the acceptance rate estimation in App. B.1. However, we highlight that the effectiveness of SD is limited by the computing resources available. Using a draft window exceeding the capacity for parallel inferences that the server can manage degrades the performance, which we show empirically later in Sec. 5. Consequently, it is essential to carefully *select* the draft window size for each request, leading to our proposed method outlined next.

3.2 Optimal Draft Token Selection

We first define a set of other notations used throughout our paper. We consider a specific LLM inference service provider with a limited capacity C, which represents the maximum number of parallel inferences its computing resources can perform. The capacity depends on the server configurations of the service provider in practice. At each time step, the server processes a batch of N requests $r_1, r_2, \cdots, r_N$, each with a partially complete sequence $S_{i,t_i} = (d_{i,1}, \ldots, d_{i,t_i})$ where t_i represents the number of tokens verified/served so far for request r_i. We allow a variable draft window size k_i for each request r_i. The draft model $\mathcal{S}$ drafts a set $\mathcal{D} \coloneqq \{(i,t)|i \in [N], t \in [t_i + k_i]\}$ such that $|\mathcal{D}| = \sum_{i=1}^{N} k_i = C$. For each $(i,t) \in \mathcal{D}$, we send $S_{i,t}$ to have its *last token* verified by $\mathcal{M}$. We aim to optimally choose the set $\mathcal{D}$ at each time step to maximize the performance of the server in terms of generation throughput, which we define below.

Per-step Throughput. For each step of SD, we are mainly concerned with maximizing the per-step throughput, i.e., the number of tokens served at each time step. Mathematically, let $\mathbf{1}_{i,t}$ be an indicator variable representing whether the last token of $S_{i,t}$ is accepted, let τ_{step} be the time per step. The per-step throughput is then defined as

$$\mathcal{G}_{\text{step}} \coloneqq (\mathbb{E}[\textstyle\sum_{(i,t)\in\mathcal{D}} \mathbf{1}_{i,t}] + N)/\tau_{\text{step}} \,.$$

Note that at least one token is always generated by SD via the *bonus token* mechanism (Leviathan et al., 2023). Thus, without considering drafting time, the throughput of SD is theoretically at least as good as that of autoregressive decoding.

Total Throughput. The total throughput is calculated as the average per-step throughput over a total of T steps with a fixed τ_{step} for each step:

$$\mathcal{G} \coloneqq T^{-1} \textstyle\sum_{i=1}^{T} \mathcal{G}_{\text{step}} \,.$$

Note that it is theoretically difficult to find an optimal draft token selection strategy that maximizes $\mathcal{G}$ as the relationship between previously verified tokens and the distribution of acceptance rate for the remaining tokens is extremely complex. However, under a mild assumption on token acceptance rate, the optimality of $\mathcal{G}$ is equivalent to the optimality of $\mathcal{G}_{\text{step}}$, as explained formally in Sec. 4.2 later.

4 TETRIS: Optimal Draft Token Selection

In this section, we introduce the details of the TETRIS for batch SD and provide an analysis of its time complexity and optimality. Overall, we leverage the insight that SD suffers from a cascading failure rate in a single sequence but not across different sequences. More specifically, we distinguish between two types of tokens involved in drafting: *sequential* and *parallel*. For each request r_i, all pairs $(i,\cdot) \in \mathcal{D}$ are sequential, i.e., for all $j < k$, (i,j) must be accepted for (i,k) to be accepted as well, implying a cascade of the failure rate. On the other hand, for $i \neq j$, $(i,\cdot)$ and $(j,\cdot)$ are parallel, as the failure rate of $(i,\cdot)$ does not influence that of $(j,\cdot)$. We highlight that the distinct nature of the two modes serves as the fundamental motivation of our proposed approach for an improved $\mathcal{G}_{\text{step}}$, and consequently the total throughput $\mathcal{G}$.

4.1 Our Approach and Design

We introduce inter-dependencies among requests within a batch. We favor parallel tokens when selecting sequential tokens leads to an excessive cascading of failure rates, and *vice versa*. To achieve this, we propose to introduce a manager to actively select the best draft tokens that are most likely to be successfully verified by the target model, thus maximizing the expected number of output tokens. The manager is integrated into the speculative decoding framework and functions as an intermediary between the draft model and the target model. It operates on the draft tokens and auxiliary outputs (e.g., token distributions, hidden states) from the draft model and strategically selects those that will be sent for verification by the target model.

At each step, define $p_{i,j}$ the conditional acceptance rate of the token at index (i,j) given its corresponding prefix. Let $\mathcal{B}_{i,j} \coloneqq (i, j, \prod_{t=1}^{j} p_{i,t})$ be the tuple containing token indices and the probability of all selected tokens in row i up to j being accepted. Instead of simply selecting a fixed window of draft tokens for verification, we *greedily* look for tokens with the highest cumulative acceptance rate $\prod_{t=1}^{j} p_{i,t}$ (and not the standalone acceptance rate $p_{i,j}$). We let the draft model propose the *extra draft tokens* beyond the server capacity and then select a set $\mathcal{D}^*$ of tokens such that it maximally utilizes the compute resource by ensuring $|\mathcal{D}^*| = C$. This process dynamically allocates longer draft windows for requests with "easy" tokens and shorter windows for "hard" ones, reducing resource wastage while sufficiently leveraging speculation, as illustrated in Fig. 1. TETRIS is outlined in Alg. 1.

Algorithm 1 TETRIS

Input: draft $\mathcal{B}$, batch size N, capacity C
Initialize $\mathcal{D}^* \leftarrow \{\}$, $\mathcal{H} \leftarrow$ Heap()
$Z \leftarrow$ InitArray$(size = N, value = -1)$
for $i \in [N]$ **do**
  $\mathcal{H}$.insert$(\mathcal{B}_{i,0})$
end for
repeat
  `// Dequeue the most probable`
  $(i, j, p_{ij}) = \mathcal{H}$.extractMax()
  $\mathcal{D}^* = \mathcal{D}^* \cup \{(i,j)\}$
  `// Record the row-wise frontier`
  $Z[i] = j$
  `// Enqueue new candidates`
  $\mathcal{H}$.insert$(\mathcal{B}_{i,j+1})$
until $|\mathcal{D}^*| = C$
return $\mathcal{D}^*$

4.2 Analysis

We now present our theoretical results, which show the per-step and global optimality of TETRIS.

Theorem 1 (Per-step Optimality of TETRIS)**.** *In the absence of drafting time, given the true acceptance rate $\alpha_{i,j}$ of each draft token (i, j), Alg. 1 produces the optimal per-step throughput defined in Sec. 3.*

The proof is delayed to App. A.1. While we have established the local optimality of TETRIS in Theorem 1, such local optimality does not trivially generalize to maximizing total throughput. Nevertheless, we show, in Theorem 2, that TETRIS is optimal in a slightly simpler scenario that retains sufficient complexity of interest.

Assumption 1. *$\forall j$, all tokens in the j-th sequence have an identical acceptance rate denoted as α_j.*

Theorem 2 (Global Optimality of TETRIS under Assumption)**.** *Under Assumption 1, in the absence of drafting time,* TETRIS *searches for the optimal $\mathcal{G}$ under the same capacity. Morever, if $\alpha_1 = \alpha_2 = \cdots = \alpha_N$,* TETRIS *has the same $\mathcal{G}$ as standard batched speculative decoding.*

The proof is delayed to App. A.3. Overall, we established both per-step and global optimality of TETRIS under theoretical assumptions. Similar assumptions are commonly made in the literature (Leviathan et al., 2023; Liu et al., 2024d) to enable theoretical insights (more details in App. B.3), and TETRIS demonstrates strong empirical performance even when this assumption is violated, as we show later in Sec. 5. In practice, the drafting time can be hidden with appropriately designed pipeline (Liu et al., 2024c; Wang et al., 2024b) which parallelizes the execution of the draft model and the target model.[1] The true acceptance rates are inaccessible in practice, we thus rely on surrogate measures and show their empirical effectiveness, which we will discuss next.

4.3 Practical Implementations

The acceptance rate of a draft token depends on $\max(p_{\mathcal{M}}(d_i)/p_{\mathcal{S}}(d_i), 1)$. However, the TETRIS manager does not have access to $p_{\mathcal{M}}(\cdot)$ before verification. In practice, we use the draft model's output probability as a surrogate of the token acceptance rate (Kim et al., 2023; Zhang et al., 2024). We show in Sec. 5 that this surrogate empirically results in strong performance. While prior works such as EAGLE-2 (Li et al., 2024b) and MDSD (Hu et al., 2025) have adopted greedy token selection based on draft model probabilities on a single request, TETRIS' greedy algorithm operates at the batch level to optimize resource utilization across multiple requests, where we defer a more detailed discussion to App. B.2. Additionally, while we theoretically show that Alg. 1 achieves a time complexity of $\mathcal{O}(C \log N)$ (see App. A.2), we can additionally leverage the parallelism of GPU to achieve empirical negligible overhead of using TETRIS (< 0.3ms compared to the average draft time per token of > 2.5ms) via the `scatter_max` operation directly implemented on GPU. Lastly, the autoregressive token drafting can also be parallelized across requests. Hence, drafting a batch of requests with a common window size of k tokens takes the same time as a single request in practice.

5 Experiments

We evaluate the effectiveness and efficiency of TETRIS against baseline methods. We first validate the necessity of dynamic draft token selection and improvement of token acceptance with TETRIS in Sections 5.1 and 5.2. Then, we show the empirical end-to-end speedup in Sec. 5.3. We also discuss the potential further improvement in empirical results with the future implementation of speculative decoding pipelines in Sec. 5.4. Our code is available at `https://github.com/ZhaoxuanWu/Tetris`.

Settings. We perform experiments on target models of various parameter sizes, including *Vicuna-33B-v1.3*, *Llama-3.1-70B-Instruct*, and *Llama-3.1-405B-Instruct*. We use *Vicuna-68M* and *Llama-3.2-1B-Instruct* as their respective draft models. Depending on the size of the models, different server configurations and tensor parallel sizes are adopted, detailed in Tab. 1. TETRIS is evaluated for generation of answer completion for questions extracted from ShareGPT (Anon, 2023), Chatbot Arena (Zheng et al., 2023), Domain Tough Ques-

Table 1: Server and model configurations. TP indicates the tensor parallel size used for model serving.

Setting	Draft Model (TP)	Target Model (TP)	GPU (VRAM)
1	Vicuna-68M (1)	Vicuna-33B (4)	4×L40 (180G)
2	Llama-1B-FP8 (1)	Llama-70B (8)	8×L40 (360G)
3	Llama-1B-FP8 (1)	Llama-405B-FP8 (8)	8×H100 (640G)

[1] Although, they have yet been integrated in popular battle-tested model serving frameworks such as `vLLM` (Kwon et al., 2023) and `SGLang` (Zheng et al., 2024) as of this writing.

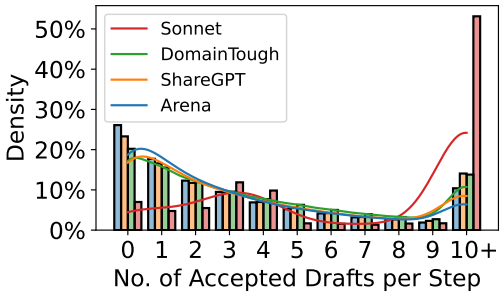


Figure 2: The distribution of the number of accepted tokens per speculative decoding step for various tasks.

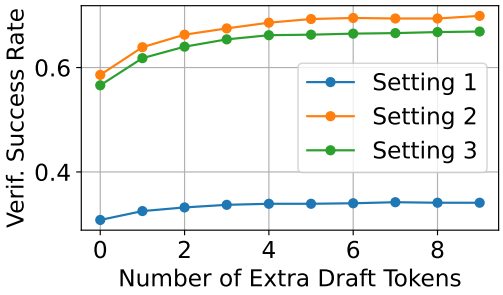


Figure 3: Change in VSR as the number of *extra draft tokens* increases. Base draft length k is set to 4, results for other k's are in App. C.2.

tions (YAV-AI, 2024), and synthetic tasks generated from Shakespeare's The Sonnet. The standard speculative decoding (SD) (Leviathan et al., 2023) and dynamic speculative decoding (DSD) (Liu et al., 2024d) are baseline methods that we compare to. We vary the drafting window sizes, allowing up to 3 extra draft tokens for TETRIS while keeping the same number of tokens sent for verification by the target model (i.e., fixing the inference capacity) for fair comparison to baseline methods. TETRIS is implemented in vLLM (Kwon et al., 2023).

5.1 Variations in Draft Quality

We begin by emphasizing the importance of setting an appropriate draft window size. Using Setting 2, we collect the oracle optimal draft window size to adopt for each SD step. Notably, the results in Fig. 2 show flat curves with long-tail distributions for various datasets, revealing significant variations in optimal window size per step. This diversity highlights the potential suboptimality of a fixed draft window, as it fails to adapt to the inherent characteristics of the draft-target model combination or a batch of sequences. By tailoring the draft token selection in a batch, TETRIS is expected to achieve higher efficiency and better alignment with the model's token acceptance patterns, hence improving overall performance.

5.2 Effect of Extra Draft Tokens

Having extra draft tokens provides TETRIS with greater flexibility in selecting which draft tokens to send for verification. To empirically show this effect, we define the verification success rate (VSR),

$$VSR = \frac{\textit{Accepted tokens}}{\textit{Tokens sent for verification}}, \quad (1)$$

which measures the quality of the draft tokens selected by TETRIS. Fixing the total number of tokens sent for verification, we show in Fig. 3 that increasing the number of extra draft tokens consistently increases the VSR metric across all settings. This finding confirms the effectiveness of TETRIS's strategy for draft token selection utilizing extra draft tokens. It also validates the empirical usefulness of the draft model's output probabilities as a surrogate of the selection criteria, as stated in Sec. 4.3.

5.3 Evaluation of TETRIS

To evaluate the effectiveness of TETRIS, we perform comprehensive experiments on various datasets and report metrics, including the total throughput and end-to-end latency. We compare to standard SD and DSD. Throughout the experiments, we maintain a consistent system load of 64 batched requests to ensure consistency, reproducibility, and fairness in comparisons. Note that all experiments include drafting time.

Total Throughput. We measure the performance of a speculative decoding method using the total throughput, which includes both accepted draft tokens by the target model and the bonus tokens, which make up the final completion. As shown in Fig. 4, TETRIS achieves up to approximately 5.25% improvement in terms of total throughput compared to the best baseline, depending on the draft-target setting and the nature of the task performed. The maximum gap between TETRIS and standard SD is up to 9.27%. Importantly, TETRIS consistently outperforms the standard SD and DSD across all settings of the draft window sizes. This shows the robustness of TETRIS to different hyperparameter choices. Additionally, it is evident that having more speculative tokens (i.e., a larger draft window size) does not always improve the performance, as having too many parallel executions of the target model exceeding the servers' parallel

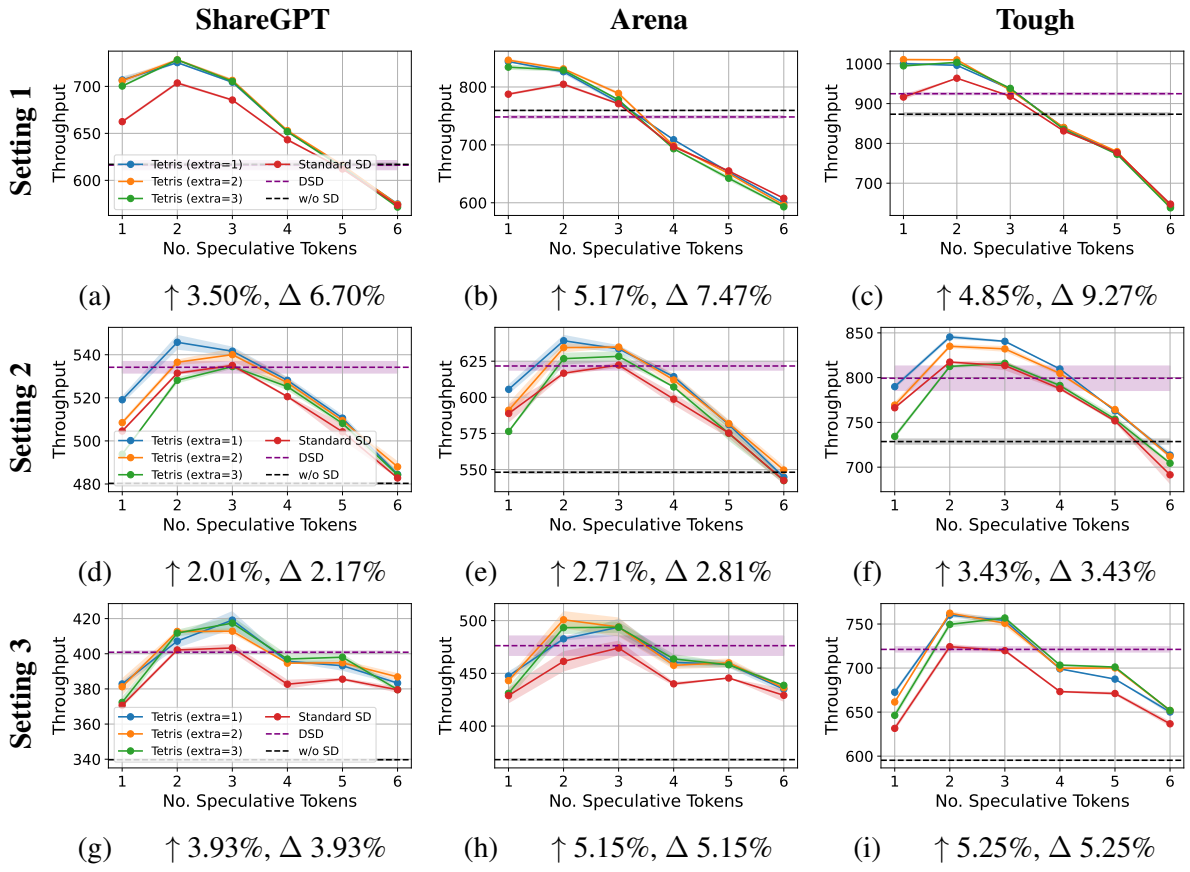


Figure 4: Throughput comparison for various methods across experimental settings. ↑ indicates the improvement over the best baseline method. Δ indicates the maximum gap between TETRIS and standard SD. The reported numbers reflect the mean and standard deviation over 3 independent trials.

inference capacity degrades performance.

Empirically, we observe that TETRIS achieves optimal performance when the number of extra draft tokens is set to 1 or 2. These results are partly attributed to the current sequential draft-target implementation for the speculative decoding pipeline, as more extra draft tokens take time to generate autoregressively. Remarkably, this pipeline can be better designed to amplify the benefit of TETRIS, which we defer the discussion to Sec. 5.4. Moreover, while DSD is expected to outperform standard SD, we note that it is not always the case in empirical experiments. This behavior may result from the difficulty of accurately estimating the conditional token acceptance rate in practice[2] and the quality of the fitted latency prediction model.

End-to-end Latency. We also measure the end-to-end latency of each request, defined as the time from sending the request to receiving the final response from the `vLLM` server on the client side.

[2]Inaccurate conditional acceptance rate estimation results in inaccurate calculation of expected generation token counts.

Table 2: Improvement in end-to-end latency. Refer to Fig. 4 for definitions of ↑ and Δ. The reported numbers reflect the mean over 3 independent trials.

Setting	ShareGPT		Arena		Tough	
	↑	Δ	↑	Δ	↑	Δ
1	3.42%	6.05%	5.30%	6.30%	5.47%	9.32%
2	2.65%	2.70%	3.86%	3.86%	3.65%	3.65%
3	3.51%	4.52%	6.13%	6.13%	4.49%	4.68%

This metric measures the average latency of the speculative decoding system in finishing completions, which can affect user satisfaction. We summarize the results in Tab. 2 and defer the figures to App. C.3. Overall, TETRIS achieves up to 6.13% improvement in latency as compared to the best baseline and up to 9.32% improvement against standard SD.

5.4 Potentially Parallelized Pipeline

We implement TETRIS to work with the `vLLM` library, one of the most efficient frameworks for LLM inference (Kwon et al., 2023). `vLLM` adopts a

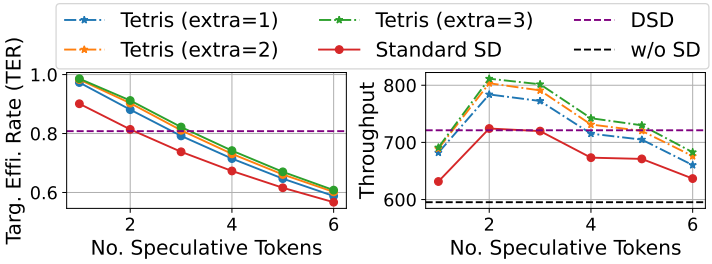


Figure 5: Left: Baseline comparisons for *TER* in different speculative configurations. Right: *Projected* $\hat{\mathcal{G}}_k^{(TER)}$ plot for TETRIS with baselines.

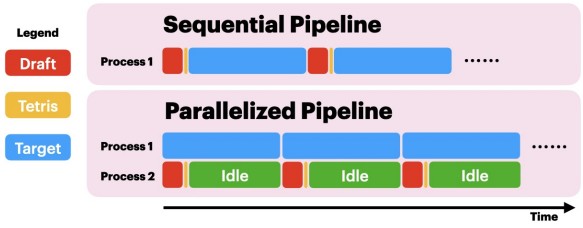


Figure 6: Parallelized pipeline for speculative decoding, where the draft model and TETRIS runtime can be hidden entirely through parallelization.

sequential pipeline for speculative decoding, where the target model runs sequentially after the draft model finishes generating draft tokens. As illustrated in Fig. 6, TETRIS is integrated between the draft and target models. However, in such a sequential pipeline, TETRIS cannot fully realize its potential as the extra draft tokens incur additional computational time.

Recent works such as Minions (Wang et al., 2024b) and PEARL (Liu et al., 2024c) have started exploring the benefits of a parallelized pipeline with two processes concurrently running the draft and target models as illustrated in Fig. 6. Given that the draft model runs significantly faster than the target model, the draft time, as well as the time to run our TETRIS, can be hidden entirely in the parallelized pipeline. Moreover, the idle time (marked in green) of Process 2 between steps can be utilized to draft more extra tokens of TETRIS or to run more complex algorithms.

Under the constraint of sequential pipelines in vLLM, we instead adopt an alternative performance metric that better captures the potential advantages of TETRIS in parallelized pipelines. We use the target efficiency rate (TER) defined as follows,

$$TER = \frac{\textit{Accepted tokens + Bonus tokens}}{\textit{Max possible number of tokens if all accepted}} . \quad (2)$$

As *TER* measures the efficiency of target model verifications and is unaffected by the drafting process and TETRIS runtime, it provides an accurate indication of the net benefit of TETRIS. In Fig. 5, we demonstrate a case study for Setting 3 on Tough dataset: The improvement of *TER* is first calculated from the left figure, and is then used to compute the *projected throughput* $\hat{\mathcal{G}}_k^{(TER)}$, following

$$\hat{\mathcal{G}}_k^{(TER)} = \mathcal{G}_{\text{SD},k} \times \frac{(TER_{\text{TETRIS},k} - TER_{\text{SD},k})}{TER_{\text{SD},k}} ,$$

where k is the number of speculative tokens (i.e., draft window size) and $\mathcal{G}$ represents throughput. Consequently, using TETRIS is *projected* (i.e., not realized in the current implementation) to achieve 12.04% improvement for this setting under parallelized pipeline. The full results are shown in Tab. 3 and the figures are shown in App. C.4.

Table 3: Projected throughput $\hat{\mathcal{G}}^{(TER)}$ improvement based on *TER* metric improvement, realizable under a parallelized speculative decoding pipeline.

Setting	Dataset	$\mathcal{G}\uparrow$	$\hat{\mathcal{G}}^{(TER)}\uparrow$
1	ShareGPT	3.50%	9.70%
	Arena	5.17%	7.79%
	Tough	4.85%	8.92%
2	ShareGPT	2.01%	11.70%
	Arena	2.71%	11.17%
	Tough	3.43%	11.91%
3	ShareGPT	3.93%	11.67%
	Arena	5.15%	10.53%
	Tough	5.25%	12.04%

5.5 Ablation Study

Robustness to Variations in Draft Quality. We artificially introduce additional variations in draft quality by mixing datasets of different difficulty levels. We create synthetic prompts designed for models to repeat lines from a poem named Sonnet. Since Sonnet is relatively easy for the small draft model, it achieves a high rate of successful verification by the target model. We then construct

Table 4: TETRIS improvement in throughput for ablation study of robustness to variations in draft quality.

Setting	**Sonnet**	**Tough**	**Mix**
1	2.46%	4.85%	4.12%
2	-0.81%	3.43%	3.48%
3	2.07%	5.25%	4.24%

a new dataset, Mix, by randomly mixing Sonnet and a more challenging dataset, Tough, in equal proportions. As shown Tab. 4, the performance improvement of TETRIS over the best baseline suffers only a marginal or no decline, indicating its robustness to substantial variations in draft quality.

Extension to Medusa. The Medusa model generates multiple subsequent draft tokens using a single forward pass (as opposed to autoregressive generation) through multiple decoding heads (Cai et al., 2024). Leveraging Medusa, it is possible to generate extra draft tokens for TETRIS at minimal marginal computational cost. We show in App. C.5 that integrating TETRIS to Medusa achieves a 3.19% improvement in total throughput.

Other Ablations. We also include ablations on TETRIS's improvement in verification success rate (VSR) in App. C.6, and the effect of batch size on the performance in App. C.7.

6 Conclusion and Future Work

In this paper, we study the problem of optimizing batch speculative decoding to maximize throughput in multi-request settings, such as those faced by model service providers. To this end, we propose TETRIS, a novel method that efficiently selects optimal draft tokens for the LLM verification in log-linear time. We have theoretically shown that, in the absence of drafting time, TETRIS achieves optimal throughput both at each decoding step and globally under reasonable assumptions about token acceptance rates. Our empirical results further validate that TETRIS consistently outperforms standard speculative decoding and existing dynamic draft window selection methods, even when accounting for the extra time required for drafting extra tokens. These results highlight the potential of TETRIS to improve inference efficiency in real-world model service deployments. A key future direction is adapting TETRIS to tree decoding, a notable feature in recent advancements in speculative decoding (Cai et al., 2024; Li et al., 2024a,b). Another interesting direction to explore is to design better draft token selection techniques. Insights can be derived from existing works on data selection (Lin et al., 2024; Zhou et al., 2023; Xu et al., 2024; Zhou et al., 2024b) that find important data points. Similarly, further research can investigate how to leverage the probability distribution between the draft model and the target model to improve the selection efficiency.

7 Limitations

In this paper, our empirical experiments only demonstrate results using the current sequential speculative decoding pipeline implemented on `vLLM`. That is, the target model stays idle while waiting for draft tokens from the draft model. Consequently, the performance improvement of TETRIS is heavily dependent on the trade-off between the additional runtime required to generate extra draft tokens and the gain in token acceptance achieved through TETRIS. Such trade-off limits the practical effectiveness of TETRIS, especially when a slow draft model is required. We anticipate that future implementations of a parallelized pipeline could potentially reveal greater speedups with TETRIS. However, we have not yet integrated such features into `vLLM` for testing in empirical experiments.

Acknowledgments

This research is supported by the National Research Foundation (NRF), Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme. The Mens, Manus, and Machina (M3S) is an interdisciplinary research group (IRG) of the Singapore MIT Alliance for Research and Technology (SMART) centre.

References

Sudhanshu Agrawal, Wonseok Jeon, and Mingu Lee. 2024. Adaedl: Early draft stopping for speculative decoding of large language models via an entropy-based lower bound on token acceptance probability. In *NeurIPS Workshop on Efficient Natural Language and Signal Processing*.

anon8231489123 Anon. 2023. Sharegpt dataset. https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024.

Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proc. ICML*, pages 5209–5235.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yu-Hsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. 2024. Sequoia: Scalable and robust speculative decoding. In *Proc. NeurIPS*, pages 129531–129563.

Yunfei Cheng, Aonan Zhang, Xuanyu Zhang, Chong Wang, and Yi Wang. 2024. Recurrent drafter for fast speculative decoding in large language models. *arXiv:2403.09919*.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and et al. 2024. Deepseek-v3 technical report. *arXiv:2412.19437*.

Fireworks AI. 2025. Fireworks ai: High-performance inference platform. https://fireworks.ai/. Accessed: 2025-02-10.

Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. 2024. Efficient LLM scheduling by learning to rank. In *Proc. NeurIPS*, pages 59006–59029.

Zhengmian Hu, Tong Zheng, Vignesh Viswanathan, Ziyi Chen, Ryan A. Rossi, Yihan Wu, Dinesh Manocha, and Heng Huang. 2025. Towards optimal multi-draft speculative decoding. In *Proc. ICLR*.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. 2024. Specdec++: Boosting speculative decoding via adaptive candidate lengths. *arXiv:2405.19715*.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. 2023. Speculative decoding with big little decoder. In *Proc. NeurIPS*, pages 39236–39256.

Taehyeon Kim, Ananda Theertha Suresh, Kishore A Papineni, Michael Riley, Sanjiv Kumar, and Adrian Benton. 2024. Accelerating blockwise parallel language models with draft refinement. In *Proc. NeurIPS*, pages 34294–34321.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proc. SOSP*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proc. ICML*, pages 19274–19286.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proc. EMNLP*, pages 7421–7432.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proc. ICML*, pages 28935–28948.

Xiaoqiang Lin, Xinyi Xu, Zhaoxuan Wu, See-Kiong Ng, and Bryan Kian Hsiang Low. 2024. Distributionally robust data valuation. In *Proc. ICML*.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. 2024a. Kangaroo: Lossless self-speculative decoding for accelerating LLMs via double early exiting. In *Proc. NeurIPS*, pages 11946–11965.

Jiachen Liu, Zhiyu Wu, Jae-Won Chung, Fan Lai, Myungjin Lee, and Mosharaf Chowdhury. 2024b. Andes: Defining and Enhancing Quality-of-Experience in LLM-Based Text Streaming Services. *arXiv:2404.16283*.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, and Winston Hu. 2024c. Parallel speculative decoding with adaptive draft length. *arXiv:2408.11850*.

Xiaoxuan Liu, Cade Daniel, Langxiang Hu, Woosuk Kwon, Zhuohan Li, Xiangxi Mo, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. 2024d. Optimizing speculative decoding for serving large language models using goodput. *arXiv:2406.14066*.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. 2024e. Online speculative decoding. In *Proc. ICML*, pages 31131–31146.

Jonathan Mamou, Oren Pereg, Daniel Korat, Moshe Berchansky, Nadav Timor, Moshe Wasserblat, and Roy Schwartz. 2024. Dynamic speculation lookahead accelerates speculative decoding of large language models. *arXiv:2405.04304*.

Nestor Maslej, Loredana Fattorini, Raymond Perrault, Vanessa Parli, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, and Jack Clark. 2024. The AI index 2024 annual report. Technical report, AI Index Steering Committee, Institute for Human-Centered AI, Stanford University.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proc. ASPLOS*, page 932–949.

Replicate. 2025. Replicate: Run AI with an API. https://replicate.com/. Accessed: 2025-02-10.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy

Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *arXiv:2308.12950*.

Hyun Ryu and Eric Kim. 2024. Closer look at efficient inference methods: A survey of speculative decoding. *arXiv:2411.13157*.

Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E. Gonzalez, and Ion Stoica. 2024. Fairness in serving large language models. In *Proc. OSDI*.

Benjamin Spector and Chris Re. 2023. Accelerating llm inference with staged speculative decoding. *arXiv:2308.04623*.

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. 2024a. Opt-tree: Speculative decoding with adaptive draft tree structure. *arXiv:2406.17276*.

Siqi Wang, Hailong Yang, Xuezhu Wang, Tongxuan Liu, Pengbo Wang, Xuning Liang, Kejie Ma, Tianyu Feng, Xin You, Yongjun Bao, Yi Liu, Zhongzhi Luan, and Depei Qian. 2024b. Minions: Accelerating large language model inference with aggregated speculative execution. *arXiv:2402.15678*.

P. Wieder, J.M. Butler, W. Theilmann, and R. Yahyapour. 2011. *Service Level Agreements for Cloud Computing*. SpringerLink : Bücher. Springer New York.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Proc. ACL Findings*, pages 7655–7671.

Xinyi Xu, Zhaoxuan Wu, Rui Qiao, Arun Verma, Yao Shu, Jingtan Wang, Xinyuan Niu, Zhenfeng He, Jiangwei Chen, Zijian Zhou, Gregory Kang Ruey Lau, Hieu Dao, Lucas Agussurja, Rachael Hwee Ling Sim, Xiaoqiang Lin, Wenyang Hu, Zhongxiang Dai, Pang Wei Koh, and Bryan Kian Hsiang Low. 2024. Data-centric AI in the age of large language models. In *Proc. EMNLP Findings*.

YAV-AI. 2024. Llm-tough-questions dataset: A collection of complex questions across 100 domains. https://huggingface.co/datasets/YAV-AI/llm-domain-specific-tough-questions.

Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. 2024. A survey on multimodal large language models. *National Science Review*, 11(12).

Fanlong Zeng, Wensheng Gan, Yongheng Wang, Ning Liu, and Philip S. Yu. 2023. Large language models for robotics: A survey. *arXiv:2311.07226*.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2024. Draft& verify: Lossless large language model acceleration via self-speculative decoding. In *Proc. ACL*, pages 11263–11282.

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. 2023. A survey of large language models. *arXiv:2303.18223*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proc. NeurIPS (Datasets and Benchmarks Track)*, pages 46595–46623.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. Sglang: Efficient execution of structured language model programs. *arXiv:2312.07104*.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2024a. Distillspec: Improving speculative decoding via knowledge distillation. In *Proc. ICLR*.

Zijian Zhou, Xiaoqiang Lin, Xinyi Xu, Alok Prakash, Daniela Rus, and Bryan Kian Hsiang Low. 2024b. Detail: Task demonstration attribution for interpretable in-context learning. In *Proc. NeurIPS*.

Zijian Zhou, Xinyi Xu, Rachael Hwee Ling Sim, Chuan Sheng Foo, and Kian Hsiang Low. 2023. Probably approximate shapley fairness with applications in machine learning. In *Proc. AAAI*.

A Leftover Proofs

A.1 Proof of Theorem 1.

Theorem 1 (Per-step Optimality of TETRIS)**.** *In the absence of drafting time, given the true acceptance rate $\alpha_{i,j}$ of each draft token (i, j), Alg. 1 produces the optimal per-step throughput defined in Sec. 3.*

Proof. We prove it by contradiction. Let the selection of Alg. 1 be $\mathcal{D}^*$. Suppose the actual optimal solution is $\mathcal{D}' \neq \mathcal{D}^*$. Let $\tilde{\mathcal{D}} = \mathcal{D}' \cap \mathcal{D}^*$ be the overlapping tokens selected by both Alg. 1 and the actual optimal solution. Note that the tokens in each row are selected sequentially (i.e., tokens cannot be skipped in a row).

Case 1: TETRIS selects some token $d \in \mathcal{D}^* \setminus \tilde{\mathcal{D}}$ before selecting $\tilde{\mathcal{D}}$. In this case, the $\mathbb{E}[\mathbf{1}]$ of the token d is higher than the token last selected in $\tilde{\mathcal{D}}$. This suggests that the optimal selection should include d. However, it can be observed that $d \notin D'$ since otherwise $d \in \tilde{\mathcal{D}}$. This contradicts the fact that $\mathcal{D}'$ is optimal.

Case 2: TETRIS selects $\tilde{\mathcal{D}}$ first before selecting other tokens. Since Alg. 1 always selects the token with the highest $\mathbb{E}[\mathbf{1}]$, every element in $\mathcal{D}^* \setminus \tilde{\mathcal{D}}$ is larger than or equal to that in $\mathcal{D}' \setminus \tilde{\mathcal{D}}$. As such, we have $\mathbb{E}[\sum_{p\in\mathcal{D}'} \mathbf{1}_p] \leq \mathbb{E}[\sum_{p\in\mathcal{D}^*} \mathbf{1}_p]$. However, this contradicts the fact that $\mathcal{D}'$ is optimal as Alg. 1 has a higher number of accepted tokens. Therefore, Alg. 1 must be optimal.

Combining the two cases finishes the proof. □

A.2 Running Time of TETRIS

Lemma 1. *Alg. 1 achieves a time complexity of $\mathcal{O}(C \log N)$.*

Proof. Note that Alg. 1 maintains a heap. The heap is initialized with N items. Since only C pairs are selected, there are $2C$ operations of enqueue and dequeue. Following classic results of heap operation, each enqueue of dequeue operation requires $\mathcal{O}(\log C)$ time. As such, the overall time complexity of TETRIS is $\mathcal{O}(C \log N)$. □

A.3 Proof of Theorem 2.

Theorem 2 (Global Optimality of TETRIS under Assumption)**.** *Under Assumption 1, in the absence of drafting time,* TETRIS *searches for the optimal $\mathcal{G}$ under the same capacity. Morever, if $\alpha_1 = \alpha_2 = \cdots = \alpha_N$,* TETRIS *has the same $\mathcal{G}$ as standard batched speculative decoding.*

Proof. The proof of global optimality is established on Theorem 1. Since all tokens in each row have the same acceptance rate. After each step, we have the same distribution of $\mathbf{1}$ no matter what tokens are accepted, where $\mathbf{1}$ is the indicator variable of whether the token is accepted. As such, at each step, performing TETRIS is per-step optimal by Theorem 1. Moreover, since the state at each step is identical, a per-step optimal strategy is also globally optimal. □

B Additional Related Work and Discussion

B.1 Acceptance Rate

The acceptance rate plays a vital role in the effectiveness of speculative decoding. A higher acceptance rate should be paired with a larger draft window size k to achieve optimal speedup. In the typical rejection sampling setting of speculative decoding, the acceptance of draft tokens depends on the probability distributions of both the draft and target models. When the probability distribution of the draft model, $p_{\mathcal{S}}(\cdot)$, closely approximates that of the target model, $p_{\mathcal{M}}(\cdot)$, a higher number of tokens are accepted on average. Since the value of k is chosen in the drafting process, we do not have access to $p_{\mathcal{M}}(\cdot)$ and have to rely on $p_{\mathcal{S}}(\cdot)$ to estimate the acceptance rate.

Leviathan et al. (2023) derive that the expected acceptance rate is 1 minus the KL divergence between the token distributions of the draft and the target model. Hence, the acceptance rates for all draft tokens are considered constant. Liu et al. (2024d) assume uniform token acceptance behavior across diverse requests. It proposes SmartSpec, which calculates the average acceptance rate from past generation steps. Li et al.

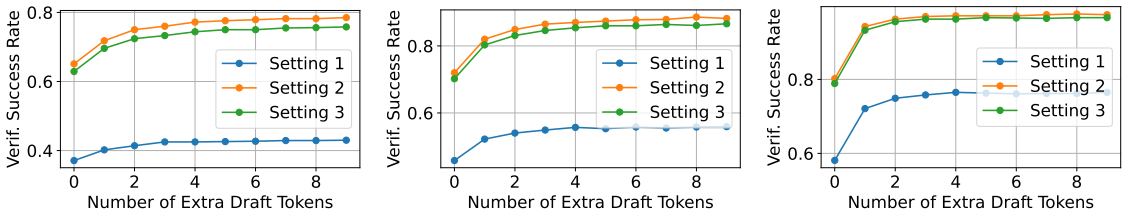
(2024a) and Wang et al. (2024a) utilize the draft model's confidence score (i.e., the output probability of each token) to estimate the acceptance rate. Chen et al. (2024) make the positional acceptance assumption so that the acceptance rate of tokens is determined solely by their position (i.e., number of tokens away) relative to the already accepted tokens. Agrawal et al. (2024) instead consider an approximate lower bound on the expected acceptance rate of a token that depends on the entropy of prediction probabilities of the draft model $p_{\mathcal{S}}(\cdot)$. Noting the acceptability of diverse tokens, especially in the real world with a high value of temperature hyperparameter, Medusa proposes to use both a hard threshold and an entropy-dependent threshold as a criterion to accept draft tokens (Cai et al., 2024). In Medusa, the first token is always accepted using greedy decoding to ensure at least one token is generated in each step.

B.2 Greedy Algorithms for Speculative Decoding

Several existing approaches in speculative decoding employ greedy algorithms to improve efficiency, though they operate at different levels and with distinct objectives. EAGLE-2 (Li et al., 2024a) and MDSD (Hu et al., 2025), for instance, utilize greedy selection strategies within single-request, multi-draft settings. EAGLE-2 achieves speedups by generating multiple branches in a draft tree, which the top candidates are then greedily selected for verification. It requires substantial GPU resources for each request and may not translate to improved throughput in batch inference due to limited total GPU capacity. MDSD, on the other hand, focuses on increasing the acceptance rate in multi-draft speculative decoding through greedy draft sampling, an objective orthogonal to our work. Our TETRIS distinguishes itself by applying a greedy algorithm at the batch level, optimizing GPU resource utilization across multiple requests rather than within a single draft tree. Due to the simplicity of greedy approaches in implementation, our method can be readily adapted to major inference frameworks (e.g., `vLLM` (Kwon et al., 2023)) with minimal empirical overhead. Our paper also demonstrates that TETRIS is complementary to existing speculative decoding frameworks, such as Medusa (Cai et al., 2024) (see Sec. 5.5 and App. C.5), by introducing a novel axis of optimization not explored by prior works.

B.3 Assumptions for Analysis

Our theoretical analysis of the global optimality of TETRIS in Theorem 2 is conditioned on Assumption 1. Similar assumptions are commonly made in the literature (Leviathan et al., 2023; Liu et al., 2024d) to analyze the length of generated outputs. Specifically, the pioneering work of Leviathan et al. (2023) assumes a constant acceptance rate α and applies a capped geometric distribution to derive the expected number of generated tokens, under a single-request setting. For a single request, this assumption is equivalent to our Assumption 1. A natural extension of the constant acceptance rate assumption to the batch inference setting would be assuming the identical acceptance rate across all requests. However, we highlight that our Assumption 1 is weaker than assuming the same constant acceptance rate across all requests, as it allows the acceptance rate to vary across requests. Moreover, even when Assumption 1 is violated, our method still demonstrates strong empirical performance as shown in Sec. 5 of the main paper.



(a) Base $k = 3$ (b) Base $k = 2$ (c) Base $k = 1$

Figure 7: Additional plots for the change in VSR as the number of *extra draft tokens* increases, supplementary to Fig. 3 where base draft length $k = 4$. Here, we provide results for $k = 1, 2, 3$.

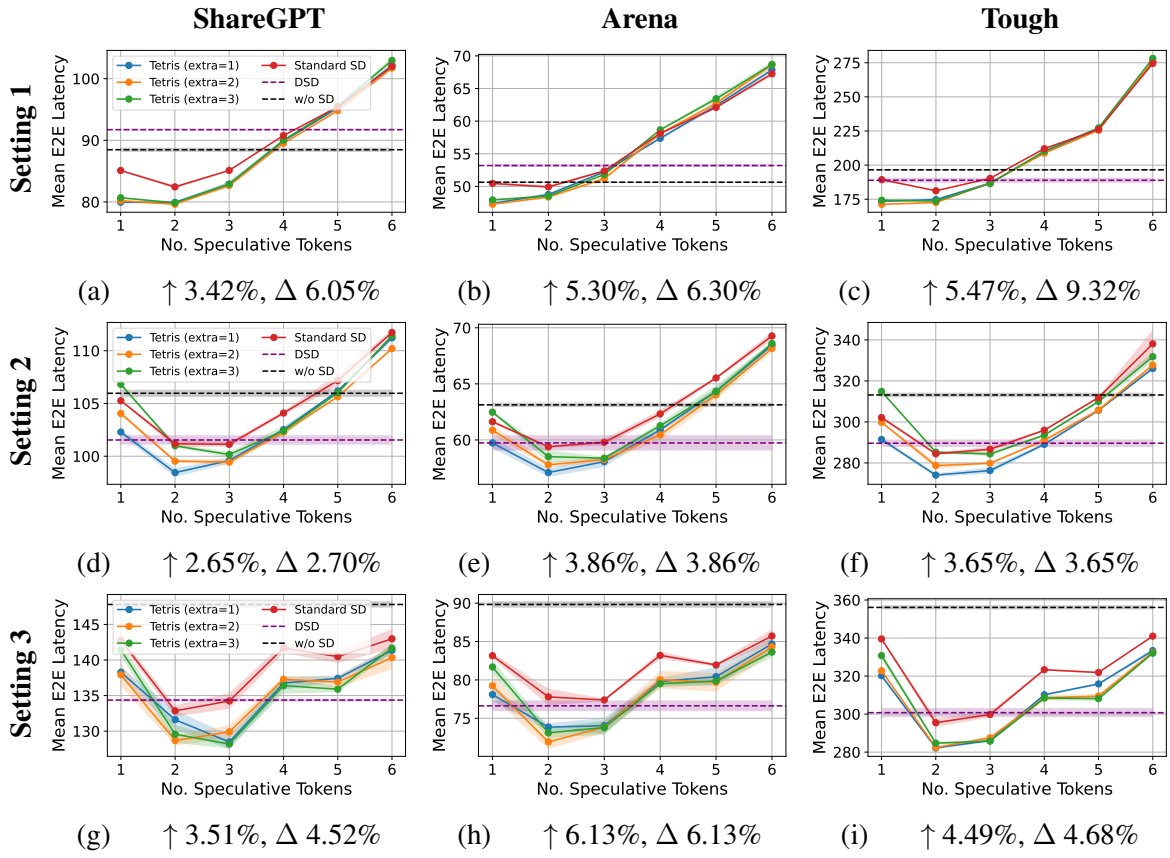


Figure 8: Mean end-to-end latency comparison for various methods across experimental settings. ↑ indicates the improvement from best baseline method. Δ indicates the maximum gap between TETRIS and standard SD. The reported numbers reflect the mean and standard deviation over 3 independent trials.

C Additional Results

C.1 Dataset License

ShareGPT (Anon, 2023): Apache license 2.0; Arena (Zheng et al., 2023): CC; Domain-specific Tough Questions (YAV-AI, 2024): MIT.

C.2 Additional Plots for Effect of Extra Draft Tokens

Fig. 3 demonstrates the benefit of adding extra draft tokens (from 0 to 9 extra tokens). The base draft length k serves only as a reference baseline and its value does not significantly impact the observed trends in Fig. 3. We set $k = 4$ based on established reasonable ranges used in existing studies (Leviathan et al., 2023; Zhang et al., 2024; Wang et al., 2024b). This choice balances the trade-off between extra compute required and additional throughput gained.

For completeness, we conducted additional experiments with $k = 1, 2, 3$ and present the results in Fig. 7. The observed trends are consistent across different values of k: VSR generally increases with extra tokens, demonstrating the effectiveness of having extra draft tokens in our TETRIS.

C.3 Plots for End-to-end Latency

We provide an extended discussion on the improvement of end-of-end latency from Sec. 5.3. In Fig. 8, we show the plots for the end-to-end latency over all speculative decoding configurations and settings used in the paper. TETRIS consistently outperforms the existing baselines and achieves up to 6.13% improvement over the best baseline and up to 9.32% maximum gap over standard SD. Therefore, TETRIS has demonstrated to effectively reduce end-to-end request latency, which is also essential for enhancing the user experience with LLM inference service providers.

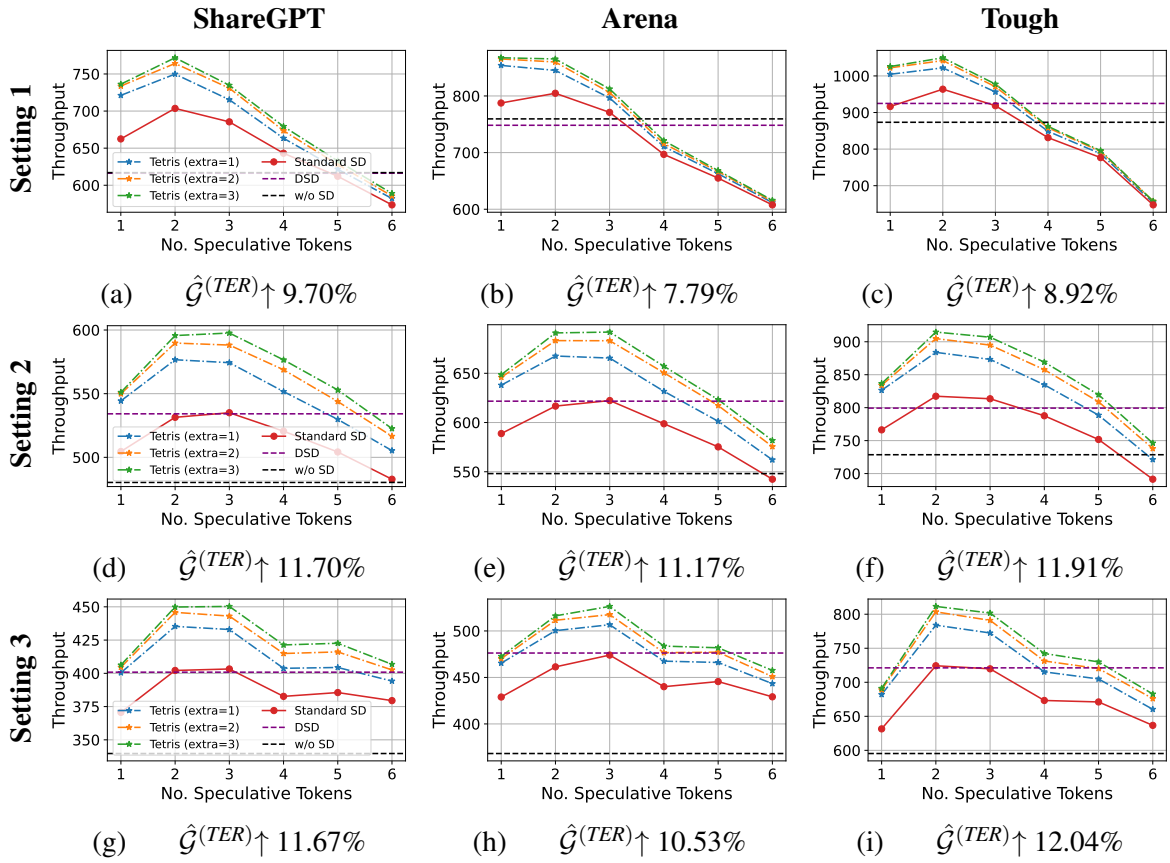


Figure 9: Mean projected throughput $\hat{\mathcal{G}}^{(TER)}$ comparison for various methods across experimental settings. $\uparrow$ indicates the improvement from the best baseline method. The reported numbers reflect the mean over 3 independent trials.

C.4 Plots for Projected Improvement based on TER

Complementary to Tab. 3, which contains the numerical results for the projected improvement of TETRIS in terms of the projected throughput $\hat{\mathcal{G}}^{(TER)}$, we also show the plots in Fig. 9 to visually illustrate the effectiveness of our method. The dotted lines for TETRIS (drawn in blue, orange, and green) represent the projected throughput calculated based on the throughput of the standard SD and also the TETRIS's improvement in terms of target efficiency rate (TER, as defined in Eq. (2)). We note that these improvement numbers are theoretically computed and are not yet realizable in empirical settings due to the lack of parallelized pipeline implementations of speculative decoding in vLLM.

C.5 Extension to Medusa

We evaluate the top-1 proposal version (i.e., only draft the most likely token for each position) of Medusa and its integration with TETRIS. As the Medusa model outputs multiple subsequent tokens in a single forward pass,[3] we leverage this feature to produce extra draft tokens for TETRIS. We show the results in Tab. 5. We achieved a throughput improvement of 3.19% as compared to the baseline Medusa. The development of such multi-token prediction models, including models like EAGLE (Li et al., 2024b) and DeepSeek-V3 (DeepSeek-AI et al., 2024) presents further potential for TETRIS to achieve greater speedups. Other improvements in engineering, including using tree-decoding and using a larger target model also potentially further boost the speedup.

[3]We use a modified implementation of Medusa in vLLM to ensure a fixed forward pass time.

Table 5: Mean total throughput (± standard deviation) for the ablation study of TETRIS extension to Medusa over three independent trials. The integration of TETRIS with Medusa further improves the total throughput.

No. Speculative Tokens	TETRIS (extra=1)	TETRIS (extra=2)	TETRIS (extra=3)	Baseline Medusa
1	591.26±0.46	590.83±8.30	586.47±3.66	572.97±1.79
2	571.05±0.80	568.82±6.52	571.95±1.06	563.94±2.95
Best	591.26	590.83	586.47	572.97

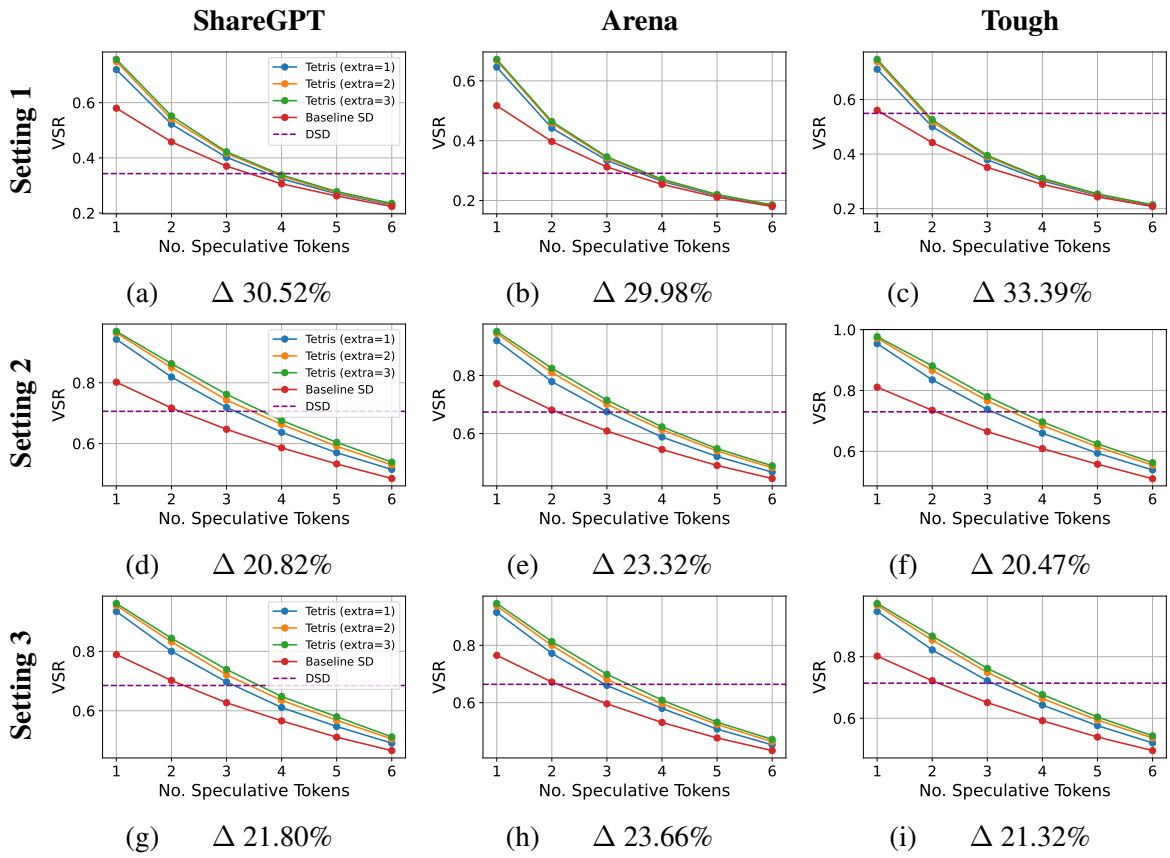


Figure 10: The verification success rate comparison for various methods across experimental settings. Δ indicates the maximum gap between TETRIS and standard SD. The reported numbers reflect the mean over 3 independent trials.

C.6 Improvement in Verification Success Rate

As an ablation study, we also illustrate the improvement of TETRIS in terms of VSR (as defined in Eq. (1)), which is an important measure of the effectiveness of speculative decoding. We show in Fig. 10 that the maximum gap between TETRIS and standard SD in terms of VSR is consistently above 20% and reaching over 30% in some instances. This validates the significant effect of TETRIS in selecting draft tokens that are most likely to be accepted by the target model without exceeding the system capacity of the server. However, it is worth noting that this improvement in VSR does not translate entirely to an increment in total throughput or a reduction in end-to-end latency. This is because the throughput in practice also depends on the running time of the draft model (especially when the speculative decoding pipeline is sequential, as discussed in Sec. 5.4), and VSR does not account for the generation of the bonus token (which takes up a portion of the generated tokens).

C.7 The Effect of Batch Size on TETRIS Performance

Theoretically speaking, a larger batch size creates more possible combinations for draft token selection by TETRIS. Therefore, TETRIS is likely to perform better in a speculative decoding server that processes a

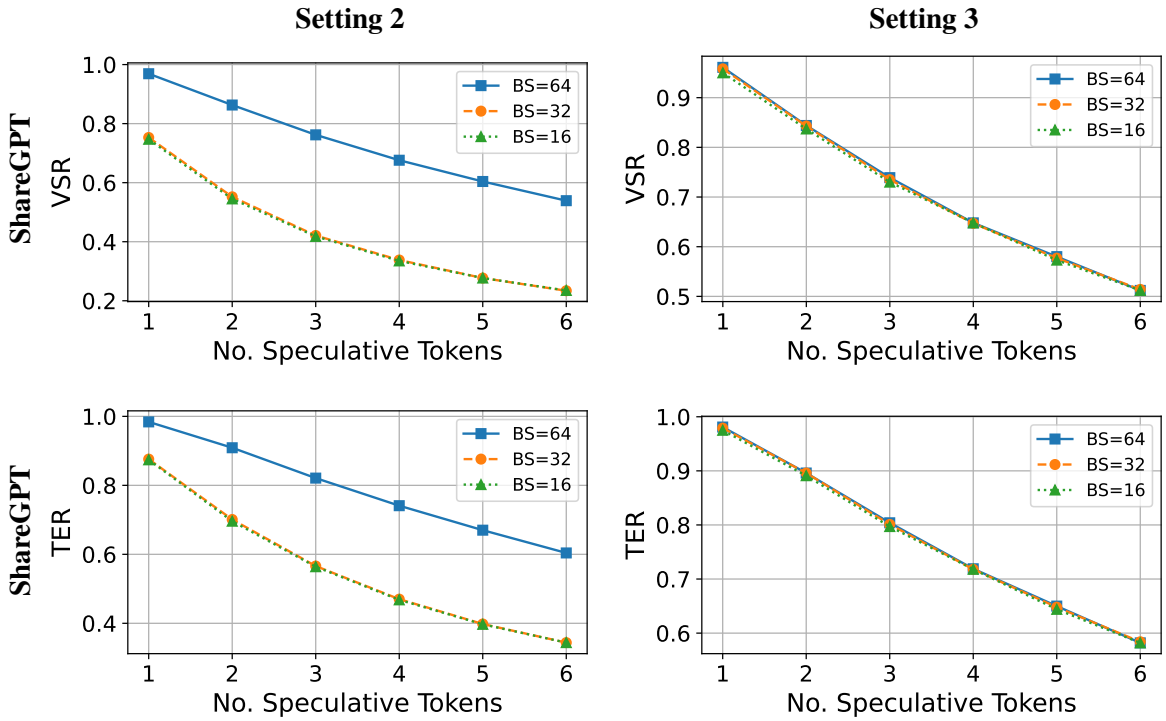


Figure 11: The change in the verification success rate (VSR) and target efficiency rate (TER) when we vary the batch size (BS) from 64 to 32 and 16. The reported numbers reflect the mean over 3 independent trials.

larger batch of requests concurrently. In Fig. 11, we show a visual illustration of the verification success rate (VSR) and target efficiency rate (TER) (as defined in Eq. (1) and Eq. (2), respectively).

In setting 2 (draft model: Llama-1B-Instruct-FP8, target model: Llama-70B-Instruct), we observe a significant increase in VSR and TER when the batch size is increased to 64. However, batch sizes of 16 and 32 have similar VSR and TER values.

In setting 3 (draft model: Llama-1B-Instruct-FP8, target model: Llama-405B-Instruct-FP8), we do not observe a significant change in VSR and TER, suggesting that the way that the batch size affects performance is highly dependent on the specific draft-target combination, too.

Overall, we expect a more significant improvement in the performance of adopting TETRIS by LLM inference service providers with larger capacities to handle a larger number of concurrent requests.

D Broader Impacts

While this research work is primarily foundational, focusing on computational performance, the resulting increase in inference speed and efficiency of Large Language Models (LLMs) could indirectly contribute to certain societal risks associated with LLMs. Making LLM inference faster and cheaper lowers the barrier to deploying these models at scale. Consequently, this could potentially accelerate the proliferation of LLM-generated content, increasing the risks of misuse such as the large-scale generation of disinformation, spam, or fake online personas, if the underlying models are deployed without adequate safeguards.

Mitigation strategies depend on responsible deployment. Developers using TETRIS should employ robust safety measures, bias detection, and content filtering for the LLMs being served. Importantly, the efficiency gains from TETRIS could be leveraged positively to make computational overhead for safety checks, alignment techniques, or bias mitigation measures more feasible during inference.