CLaSp: In-Context Layer Skip for Self-Speculative Decoding

Longze Chen[1,2*] **Renke Shan**[2,5*] **Huiming Wang**[3*] **Lu Wang**[5] **Ziqiang Liu**[1,2]
Run Luo[1,2] **Jiawei Wang**[2] **Hamid Alinejad-Rokny**[4] **Min Yang**[1†]
[1] Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
[2] University of Chinese Academy of Sciences [3] Singapore University of Technology and Design
[4] School of Biomedical Engineering, UNSW Sydney [5] Ritzz-AI
{lz.chen2, min.yang}@siat.ac.cn

Abstract

Speculative decoding (SD) is a promising method for accelerating the decoding process of Large Language Models (LLMs). The efficiency of SD primarily hinges on the consistency between the draft model and the verify model. However, existing drafting approaches typically require additional modules to be trained, which can be challenging to implement and ensure compatibility across various LLMs. In this paper, we propose **CLaSp**, an in-context layer-skipping strategy for self-speculative decoding. Unlike prior methods, CLaSp does not require additional drafting modules or extra training. Instead, it employs a plug-and-play mechanism by skipping intermediate layers of the verify model to construct a compressed draft model. Specifically, we develop a dynamic programming algorithm that optimizes the layer-skipping process by leveraging the complete hidden states from the last verification stage as an objective. This enables CLaSp to dynamically adjust its layer-skipping strategy after each verification stage, without relying on pre-optimized sets of skipped layers. Experimental results across diverse downstream tasks demonstrate that CLaSp achieves a speedup of $\mathbf{1.3}\times \sim \mathbf{1.7}\times$ on LLaMA3 series models without altering the original distribution of the generated text.

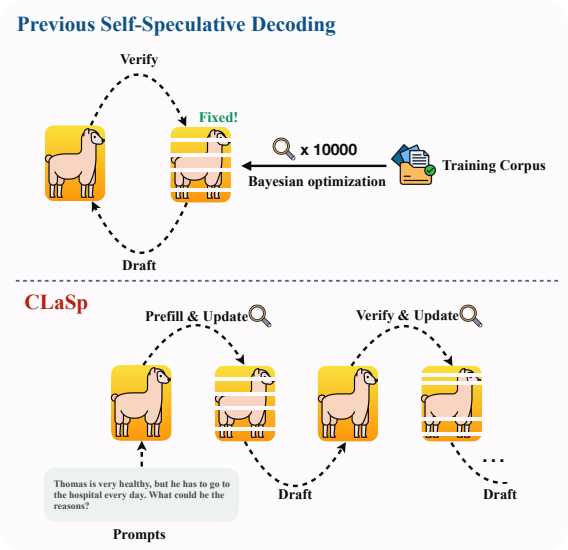


Figure 1: **Previous Self-SD method vs. CLaSp**. Compared to the previous Self-SD method, which requires costly Bayesian optimization on training dataset to select a ***fixed*** set of skipped layers, CLaSp employs a ***dynamic*** layer-skipping strategy that adjusts in real-time based on context.

1 Introduction

Transformer-based Large Language Models (LLMs) have achieved remarkable success across a wide range of natural language processing applications (Brown et al., 2020; Achiam et al., 2023). Scaling the model size and extending the context window significantly enhance performance (Kaplan et al., 2020; Anil et al., 2023; Reid et al., 2024), but also leads to a rapid increase in inference latency. This latency primarily stems from the autoregressive nature of LLMs, where model parameters must be loaded into GPU SRAM for each token generation, resulting in underutilization of computational cores during the decoding stage (Patterson, 2004; Shazeer, 2019; Agrawal et al., 2023).

Inspired by speculative execution in computer systems (Burton, 1985; Hennessy and Patterson, 2012), speculative decoding (SD) (Xia et al., 2023; Leviathan et al., 2023; Chen et al., 2023) is proposed as a lossless autoregressive decoding acceleration technique. SD accelerates autoregressive decoding by introducing an efficient draft model to pre-generate tokens, which are subsequently validated by a slower verify model in parallel. This technique significantly reduces the number

[*] Equal contribution.
[†] Corresponding author.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 31608–31618
July 27 - August 1, 2025

of forward passes required by the verify model, alleviating memory-bound inefficiencies caused by frequent parameter access. While effective, SD relies on finding or training a suitable draft model that can closely mimic the behavior of the verify model. This requirement is feasible for open-sourced model families, such as LLaMA series (Touvron et al., 2023a,b; Dubey et al., 2024; Yang et al., 2024), but becomes prohibitively difficult for specialized LLMs that lack pre-existing compatible draft model counterparts.

The difficulty lies in achieving consistency between the draft model and the verify model. For general-purpose models, lightweight modules have been proposed as substitutes for the draft model (Cai et al., 2024; Li et al., 2024b; Du et al., 2024; Liu et al., 2024). These modules are designed to avoid retraining from scratch, but they struggle to generalize across diverse tasks and contexts. As a result, their acceptance rates drop sharply when handling unseen tasks, making them unsuitable for applications requiring robust performance across varying scenarios.

Self-speculative decoding (Self-SD) (Zhang et al., 2024) addresses the challenge of compatibility by using parts of the verify model itself as a compressed draft model, bypassing the need for additional modules or training. This approach creates the draft model by sparsifying intermediate layers of the verify model, effectively skipping certain computations. Similar to methods that require training, it also lacks robust generalization and heavily relies on an time-consuming Bayesian optimization process. SWIFT (Xia et al., 2024a) extends Self-SD by dynamically optimizing skipped layers as the number of user requests increases, but its effectiveness diminishes when handling sparse or unique task data.

To bridge this gap, we propose **CLaSp**, a dynamic in-context layer-skipping method for self-speculative decoding. Unlike existing methods, CLaSp dynamically adjusts the skipped layer set at each decoding step based on the current context, eliminating the need for pre-optimization or retraining (see Figure 1). Our approach leverages the observation of slowly changing embeddings across layers (Liu et al., 2023) and employs a dynamic programming algorithm to identify the optimal skipped layers with minimal additional latency. By using the complete hidden states from the last verification step as ground truth, CLaSp predicts and adjusts the draft model's sparsity in real-time, achieving high acceptance rates while maintaining acceleration benefits.

We evaluate CLaSp on the LLaMA3 series models using Spec-Bench (Xia et al., 2024b), a comprehensive benchmark for speculative decoding across diverse scenarios. CLaSp achieves $1.3\times \sim 1.7\times$ wallclock time speedup compared to conventional autoregressive decoding while preserving the original distribution of generated text. Our contributions are summarized as follows:

- We introduce CLaSp, a self-speculative decoding framework that dynamically adjusts the layer-skipping strategy based on context.
- We propose performance optimization strategies in CLaSp to fully leverage GPU parallelism, making the extra latency from layer optimization almost negligible.
- We conduct extensive experiments on Spec-Bench, showing that CLaSp consistently achieves $1.3\times \sim 1.7\times$ speedup without training, and provide a detailed analysis of its key hyper-parameters.

2 Related Work

Speculative Decoding. Speculative decoding (Xia et al., 2023; Leviathan et al., 2023; Chen et al., 2023) has been proposed as an effective strategy for lossless acceleration of LLM inference. Some approaches aim to reduce the high cost of training from scratch by introducing lightweight modules as draft model. Medusa (Cai et al., 2024) trains multiple decoding heads to predict the next n tokens in parallel. EAGLE and EAGLE-2 (Li et al., 2024b,a) integrate a lightweight plug-in (a single transformer decoder layer) to existing LLMs. Glimpse Draft Model (Du et al., 2024) reuses the verify model's KV cache to generate candidate tokens that are more likely to be accepted by the verify model. However, these approaches rely heavily on pre-trained modules optimized for specific contexts, making them less effective for tasks with unseen data.

Retrieval-based methods, such as REST (He et al., 2024) and Prompt Lookup Decoding (Saxena, 2023), replace the draft model by retrieving relevant drafts from a text corpus or context based on input prompts. While these methods reduce reliance on explicit draft model models, their performance is highly sensitive to the quality and relevance of the retrieved drafts.

To address the challenges of designing compatible draft model, Self-SD (Zhang et al., 2024) and SWIFT (Xia et al., 2024a) directly leverage parts of the verify model as a compressed draft model by skipping intermediate layers of the original LLM. Triforce (Sun et al., 2024) employs a partial KV cache as the draft model and a full KV cache as the verify model, reducing inference latency by minimizing I/O operations, especially in long-context tasks. Despite these innovations, static configurations for layer skipping prevent these methods from dynamically adapting to changing task requirements or contexts, limiting their efficiency and generalizability.

To further enhance speculative decoding, tree-based attention mechanisms (Miao et al., 2024; Cai et al., 2024; Chen et al., 2024; Svirschevski et al., 2024) extend the decoding process from generating a single candidate sequence to exploring a candidate tree. By providing the verify model with multiple options for validation, these methods improve the acceptance rate of speculative decoding.

Layer-wise Sparsity. Layer redundancy in LLMs has been extensively studied, with methods such as LayerDrop (Fan et al., 2020), LayerSkip (Elhoushi et al., 2024), structured pruning (Zhang and He, 2020), SkipDecode (Corro et al., 2023), and LayerSharing (Zhang et al., 2023) demonstrating that not all layers are equally important. These methods suggest that the importance of each layer varies depending on the task and context, and that certain layers can be skipped or removed without significantly affecting model performance. However, determining which layers to skip or optimize for different downstream tasks remains a substantial challenge.

Deja Vu (Liu et al., 2023) and LISA (pan, 2024) demonstrate the potential of leveraging sparsity to accelerate LLM inference, either by exploiting context sparsity or by optimizing a subset of layers during training. Although effective, these methods rely on lossy sparsification techniques, introducing discrepancies between the sparse and original distributions. Similarly, Glavas et al. (2024) explore dynamic inference methods such as layer skipping and early exiting, which enable task-dependent acceleration but lack compatibility with speculative decoding, limiting their potential for lossless acceleration. Thus, we aim to combine the strengths of layer-wise sparsity and speculative decoding to achieve lossless acceleration across diverse tasks.

Algorithm 1: CLaSp Skip Layer Strategy

Input: Num hidden layers L, num skip layers M, hidden states $X = \{x_0, x_1, ..., x_{L-1}\}$, DecoderLayer f_i, hidden size d
Output: The optimal skipped layer set $\mathcal{S}$

$g \leftarrow \text{zeros}(L+1, M+1, d)$, $g[0,0] \leftarrow x_0$
// Dynamic programming
for $i = 1$ **to** $L+1$ **do**
  $g[i,0] \leftarrow x_i$
  $\ell \leftarrow \min(i-1, M)$
  $\mathcal{G} \leftarrow f_{i-1}(g[i-1, 1:\ell+1])$
  $\mathcal{F} \leftarrow \text{norm}(\text{cat}(\mathcal{G}, g[i-1, :\ell]))$
  $\sigma \leftarrow \mathcal{F} \cdot \text{norm}(x_i)$
  if $\sigma[:\ell] > \sigma[\ell:]$ **then**
    $g[i][1:\ell+1] \leftarrow \mathcal{G}$
  else
    $g[i][1:\ell+1] \leftarrow g[i-1, :\ell]$
  if $i \leq M$ **then**
    $g[i,i] \leftarrow g[i-1, i-1]$
$\mathcal{S} \leftarrow \text{zeros}(L)$
// Backtracking optimal skipped layer set $\mathcal{S}$
while $i > 0$ ***and*** $j > 0$ **do**
  if $g[i,j] = g[i-1, j-1]$ **then**
    $S[i-1] \leftarrow 1$
    $j \leftarrow j-1$
  $i \leftarrow i-1$
return $\mathcal{S}$

3 CLaSp

In this section, we first introduce the pipeline of CLaSp from a global perspective. Then, we explore the main challenges (§3.2) faced by CLaSp and formulate the problem of layer skipping (§3.3). Subsequently, we provide a detailed description of the CLaSp algorithm (§3.4 and §3.5) and efficiency optimization strategies (§3.6 and §3.7).

3.1 Pipeline

CLaSp can be explained as a three-stage process: (1) **Drafting**: The draft model autoregressively generates K draft tokens from the given prompt sequence $x_1, \ldots, x_i$, denoted as $x_{i+1}, \ldots, x_{i+K}$. (2) **Verification**: The verify model verifies the tokens generated during the drafting stage. This verification is performed in a single forward pass, where the LLM predicts the probability distribution for each draft token and evaluates whether they align with the full model's predictions. Once a draft token x_j is rejected, the original LLM's prediction overwrites x_j, and drafting resumes from token x_{j+1} in the next round. (3) **Layer Optimization**: Using the hidden states of the last accepted token x_j as the optimization objective, the optimal skipped layer set $\mathcal{S}^*$ is updated to guide the next round of drafting. As shown in Figure 2, before each round of drafting, the draft model can be updated to better adapt to the current context.

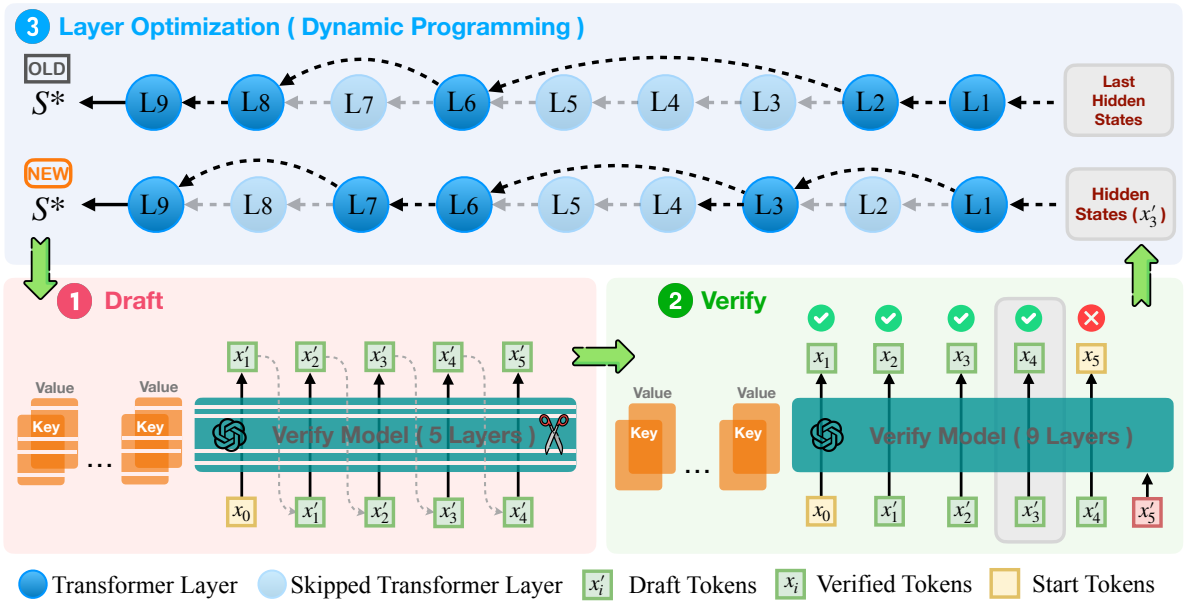


Figure 2: The overall framework of CLaSp consists of three stages: (1) Draft, (2) Verify, (3) Layer Optimization. After the Verify stage, CLaSp uses the information obtained to perform Layer Optimization, resulting in a new optimal layer skipping set $\mathcal{S}^*$. This set guides the next Draft round, repeating the entire process.

3.2 Main Challenges

Compared to previous methods, CLaSp dynamically updates the skipped layer set before each drafting step, requiring solutions to two main challenges:

(1) How to determine which layers should be skipped? This is the most critical issue addressed by CLaSp, as it directly impacts drafting quality. An ideal layer-skipping strategy must adapt to the most recent context, ensuring that the drafted tokens are more likely to be accepted by the verify model.

(2) How to reduce the additional latency caused by layer optimization? The dynamic skipping strategy inevitably introduces computational delays due to the need for repeated searches to identify the current optimal layer subset. To ensure that layer optimization does not become the primary bottleneck, minimizing these additional delays is essential for maximizing the speedup benefits.

3.3 Problem Formulation of Layer Skip

Let $\mathcal{M}_v$ be the verify model and $\mathcal{M}_d$ be the draft model obtained by skipping certain intermediate layers from the original LLM. $F_{\mathcal{M}_v}(X)$ and $F_{\mathcal{M}_d}(X)$ represent the output hidden states on the top of the last token of current input X, passing through the verify model or the draft model respectively. Our goal is to find the optimal skipped layer set $\mathcal{S}$ that minimizes the cosine similarity between $F_{\mathcal{M}_v}(X)$ and $F_{\mathcal{M}_d}(X)$:

$$\mathcal{S}^* = \arg\min_{\mathcal{S}} cosine(F_{\mathcal{M}_V}(X), F_{\mathcal{M}_D}(X)), \quad \text{s.t. } \mathcal{S} \in \{0,1\}^L \tag{1}$$

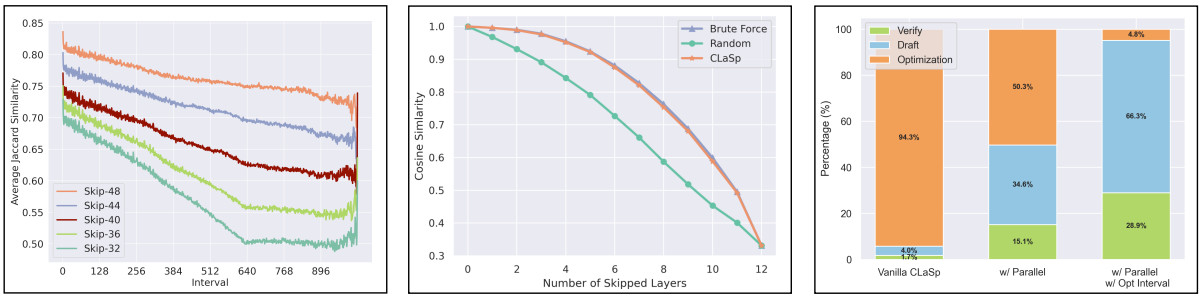
where L represents the number of transformer layers in the verify model.

3.4 Approximate Dynamic Programming

The principle behind selecting information for layer optimization is to minimize additional computational overhead by utilizing information already obtained in previous steps. In speculative decoding, we observed that the hidden states of the last accepted token after each verification step are not fully utilized. To address this, we propose leveraging this feedback information to predict the draft model configuration for the next drafting stage. Specifically, let the input tokens to a Transformer model be denoted as X, with an embedding layer that maps token indices to token embeddings h_0. The Transformer model consists of L layers, where the l-th Transformer layer performs a transformation f_l, evolving the embeddings as:

$$h_{l+1} = f_l(h_l)$$

Let $\mathcal{D}(i,j)$ represent the maximum cosine similarity between h_i and the optimal hidden state



(a) (b) (c)

Figure 3: (a) **Sparse Persistence Observation**: Skipped layer sets selected for adjacent tokens exhibit high similarity, with this similarity gradually decreasing as the token gap increases. This observation enables layer optimization on the current token to guide subsequent drafting processes. (b) **Approximate Markov Property**: Cosine similarity comparisons of hidden states obtained using Brute Force, Random, and CLaSp's dynamic programming configurations against the full forward pass demonstrate the approximate Markov property inherent to CLaSp. (c) **Efficiency Optimization Strategies**: Latency breakdown per query shows that Layer Optimization introduces only 4.8% additional delay, underscoring its negligible impact on overall latency.

$g(i, j)$ obtained by skipping j layers among the first i transformer layers. So we design a dynamic programming transition equation defined as:

$$\mathcal{D}(i, j) = \max\{cosine(h_i, g(i-1, j-1)), \\ cosine(h_i, f_{i-1}(g(i-1, j)))\} \quad (2)$$

where $cosine$ is used to calculate the cosine similarity between two vectors. The CLaSp skip layer algorithm process is shown in Algorithm 1.

3.5 Approximate Markov Property

A crucial prerequisite for dynamic programming algorithms is the "no aftereffect" property, which ensures that current decisions and state transitions are independent of previous states. However, when computing the optimal hidden states $g(i, j)$, CLaSp does not strictly satisfy the Markov property, making it theoretically impossible to find an exact optimal solution using Algorithm 1.

Fortunately, due to the favorable property of slowly changing embeddings across layers, we observe that CLaSp's approximate algorithm closely aligns with the results of a brute force search for the optimal skipped layer set. To validate this, we fixed the first and last 10 layers of the 32-layer LLaMA3-8B model and compared the outcomes of brute force search, random layer selection, and CLaSp across the remaining 12 layers.

As shown in Figure 3b, the hidden states obtained by skipping the layers selected by CLaSp exhibit remarkable consistency with those from the brute force search, demonstrating a high cosine similarity with the hidden states of the original LLM. In contrast, the results from randomly selected layers show relatively poor alignment.

These findings indicate that CLaSp approximates the Markov property effectively, allowing it to find near-optimal solutions within an acceptable error range.

3.6 Sequence Parallel

Unlike previous methods, CLaSp requires multiple layer optimizations during a single inference process. Therefore, the optimization process must be both efficient and accurate to avoid introducing additional delays while ensuring precise drafting in subsequent decoding steps. To address this, we employ parallelization strategies to minimize the additional delay caused by the dynamic programming process.

When CLaSp performs dynamic programming, the updates for $\mathcal{D}(i, j)$ and $g(i, j)$ are obtained through a double loop, resulting in a time complexity of $\mathcal{O}(LM)$. Importantly, when computing the state at (i, j), only the state at $(i-1, \cdot)$ is required. This dependency allows computations for different j values with the same i to be performed independently, enabling parallelization of the second loop.

To further reduce the GPU memory footprint, we avoid concatenating these states into a batch. Instead, we design a specialized mask matrix that enables parallelization of these states as a sequence. This approach reuses the same KV cache without duplicating it, significantly improving memory efficiency.

Models	Methods	MT-bench		WMT14		CNN/DM		NQ		GSM8K		DPR		Overall Speedup
		τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	
Greedy Setting: Temperature=0														
LLaMA-3-70B	AUTOREGRESSIVE	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00×
	SELF-SD	2.57	1.38×	4.10	1.55×	5.46	1.57×	2.60	1.42×	3.10	1.49×	3.59	1.43×	1.47×
	SWIFT	3.13	1.26×	2.90	1.27×	3.93	1.35×	3.21	1.29×	2.86	1.27×	3.31	1.26×	1.28×
	CLASP	4.55	**1.64**×	5.81	**1.69**×	7.19	**1.66**×	5.37	**1.72**×	6.77	**1.75**×	4.05	**1.56**×	**1.67**×
LLaMA-3-70B -Chat	AUTOREGRESSIVE	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00×
	SELF-SD	1.40	1.23×	2.27	1.33×	1.50	1.24×	1.59	1.26×	3.00	1.40×	2.56	1.37×	1.31×
	SWIFT	4.41	1.15×	5.54	1.27×	4.52	1.22×	4.83	1.20×	6.19	1.31×	5.97	1.33×	1.25×
	CLASP	2.61	**1.35**×	4.72	**1.51**×	3.48	**1.39**×	3.32	**1.39**×	5.28	**1.53**×	5.61	**1.54**×	**1.45**×
LLaMA-3-8B	AUTOREGRESSIVE	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00×
	SELF-SD	1.28	1.07×	1.35	1.13×	1.73	1.17×	1.45	1.13×	1.44	1.15×	2.33	1.21×	1.14×
	SWIFT	2.75	1.07×	2.51	1.09×	2.76	1.13×	2.91	1.13×	2.72	1.10×	2.96	1.11×	1.11×
	CLASP	3.68	**1.24**×	4.14	**1.23**×	6.22	**1.22**×	4.03	**1.27**×	5.26	**1.26**×	4.17	**1.22**×	**1.24**×
Non-Greedy Setting: Temperature=1														
LLaMA-3-70B	AUTOREGRESSIVE	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00×
	SELF-SD	1.64	1.23×	2.53	1.39×	3.61	1.43×	1.53	1.24×	2.01	1.33×	2.17	1.24×	1.31×
	SWIFT	2.06	1.10×	1.96	1.08×	1.97	1.09×	1.97	1.08×	1.98	1.09×	2.01	1.07×	1.09×
	CLASP	3.13	**1.49**×	3.33	**1.50**×	5.38	**1.54**×	3.56	**1.54**×	4.32	**1.59**×	2.51	**1.36**×	**1.50**×
LLaMA-3-70B -Chat	AUTOREGRESSIVE	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00×
	SELF-SD	1.15	1.14×	2.01	1.23×	1.19	1.15×	1.21	1.17×	1.97	1.34×	1.71	1.26×	1.22×
	SWIFT	2.68	0.96×	2.64	0.99×	2.67	0.98×	2.62	0.99×	2.79	1.01×	2.76	1.04×	1.00×
	CLASP	1.96	**1.28**×	3.90	**1.45**×	2.32	**1.29**×	2.28	**1.30**×	4.40	**1.47**×	4.03	**1.43**×	**1.37**×
LLaMA-3-8B	AUTOREGRESSIVE	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00	1.00×	1.00×
	SELF-SD	0.98	0.89×	1.01	0.94×	1.36	1.02×	1.09	0.92×	1.09	0.96×	1.82	1.03×	0.96×
	SWIFT	1.90	0.80×	1.92	0.85×	1.85	0.83×	1.97	0.84×	1.95	0.83×	1.90	0.80×	0.83×
	CLASP	2.62	**1.11**×	2.78	**1.08**×	4.26	**1.11**×	2.70	**1.08**×	3.76	**1.10**×	2.35	**1.02**×	**1.08**×

Table 1: Comparison between CLaSp and prior plug-and-play methods. We report the average acceptance length τ and speedup ratio under greedy (Temperature=0) and non-greedy (Temperature=1) settings. **Bold** numbers denotes the best Speedup.

3.7 Lower Optimization Frequency

CLaSp updates the optimal skipped layer set after each verification step, using the feedback from the last accepted token. However, the time cost of performing this update is comparable to that of a verification step, which introduces a bottleneck for CLaSp's inference latency. Fortunately, we observe a phenomenon we term **Sparse Persistence**: the skipped layer sets required by adjacent tokens tend to exhibit high similarity. To quantify this, we calculate the Jaccard similarity between the skipped layer sets of adjacent tokens. As shown in Figure 3a, the similarity remains high when the token distance is within a certain range and only decreases significantly as the distance between tokens increases.

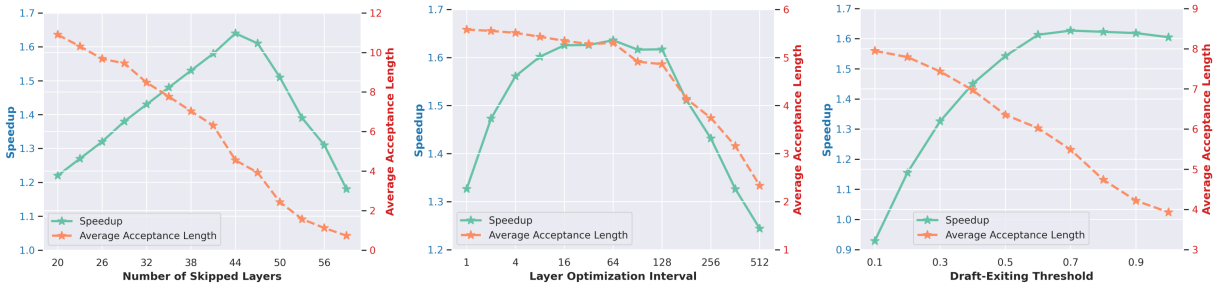
Based on this observation, we further found that the optimal skipped layer set does not change drastically after every update. This allowed us to adjust the update frequency by accumulating several verification steps before performing an update, rather than updating after every single verification step. While adopting a lower update frequency slightly reduced the average acceptance rate of draft tokens, the reduction in update latency led to a substantial improvement in the overall speedup ratio. This trade-off highlights the efficiency benefits of leveraging Sparse Persistence to optimize the layer update process.

4 Experiments

This section evaluates CLaSp across various text generation tasks to demonstrate its efficiency and effectiveness.

Model and Testbed. We evaluate CLaSp using four different sizes of LLaMA models (Dubey et al., 2024): LLaMA3-8B, LLaMA2-13B, LLaMA3-70B, and LLaMA3.1-405B. The models are deployed on NVIDIA A800 GPUs with 80GB of memory. Specifically, the 8B and 13B models are deployed on a single A800 GPU, while the 70B and 405B models utilize 2 and 8 A800 GPUs, respectively, with pipeline parallelism enabled by Accelerate (Gugger et al., 2022). All models use FP16 precision, except for LLaMA3.1-405B, which employs INT8 quantization for improved memory efficiency. Unless otherwise specified, the batch size is set to 1 for all models.

Datasets. We benchmark the performance of CLaSp on Spec-Bench (Xia et al., 2024b), a comprehensive evaluation suite covering a wide range of datasets and tasks. Spec-Bench includes six subtasks: multi-turn conversation, translation, summarization, question answering, mathematical reason-



(a) (b) (c)

Figure 4: The impact of key hyper-parameters on speedup: (a) Number of Skipped Layers; (b) Layer Optimization Interval; (c) Draft-Existing Threshold. The experiment results were all obtained using the LLaMA-3-70B model on MT-Bench.

ing, and retrieval-augmented generation. Specifically, Spec-Bench consists of 80 randomly selected instances from each of MT-bench (Zheng et al., 2023), WMT14 DE-EN, CNN/Daily Mail (Nallapati et al., 2016), Natural Questions (Kwiatkowski et al., 2019), GSM8K (Cobbe et al., 2021), and DPR (Karpukhin et al., 2020). To control the generation length across tasks, we set the maximum sequence length to 1024 tokens, following prior experimental setups (Xia et al., 2024b).

Comparison. For our main experiments, we use vanilla autoregressive decoding as the baseline, serving as the benchmark for speedup ratios (1.00x). We compare CLaSp against existing training-free layer skip methods, including Self-Speculative Decoding (Zhang et al., 2024) and SWIFT (Xia et al., 2024a). Other speculative decoding (SD) methods are excluded from the comparison as they require additional modules or extensive training, which limits their generalizability. Since the speedup ratio is hardware-dependent, all methods were evaluated on the same devices to ensure a fair comparison.

Performance Metrics. CLaSp is essentially still speculative sampling, which has been proven to enable lossless acceleration (Leviathan et al., 2023). Therefore, we focus solely on acceleration performance rather than generation quality. The following metrics are used for evaluation: **Speedup Ratio**: The actual test speedup ratio relative to vanilla autoregressive decoding, providing a direct measure of acceleration. **Average Acceptance Length** (τ): The average number of tokens generated per drafting-verification cycle, corresponding to the number of tokens accepted from the draft. This metric is independent of hardware and runtime environment, while its limitation is that it does not reflect the overhead introduced by the draft model.

4.1 Experimental Result

As shown in Table 1, we report the performance of CLaSp and prior plug-and-play methods on text generation tasks from Spec-Bench under both greedy (Temperature = 0) and non-greedy (Temperature = 1) settings. The experimental results reveal the following findings.

CLaSp demonstrates superior efficiency compared to previous methods, achieving consistent speedups of $1.3\times \sim 1.7\times$ over vanilla autoregressive decoding across various models and tasks. Prior methods relying on Bayesian optimization exhibit lower performance, particularly when data volume is limited.

CLaSp consistently improves average acceptance length, acceptance rate, and speedups. This efficiency is primarily due to CLaSp's ability to leverage model layer sparsity effectively. By skipping 50% to 60% of layers during experiments, CLaSp maintains both a high average acceptance length and acceptance rate, contributing to superior acceleration. Generally, longer acceptance lengths lead to higher speedups. However, there are cases where speedups remain low despite long acceptance lengths, as drafting additional tokens increases time spent, reducing acceptance rates and overall speedups.

The performance advantage of CLaSp is more pronounced on larger models, such as LLaMA3-70B, compared to smaller models like LLaMA2-13B and LLaMA3-8B. This suggests that CLaSp can better leverage the greater layer sparsity present in larger models, enhancing adaptability and efficiency.

Overall, the robust performance of CLaSp across different models highlights its effectiveness as a

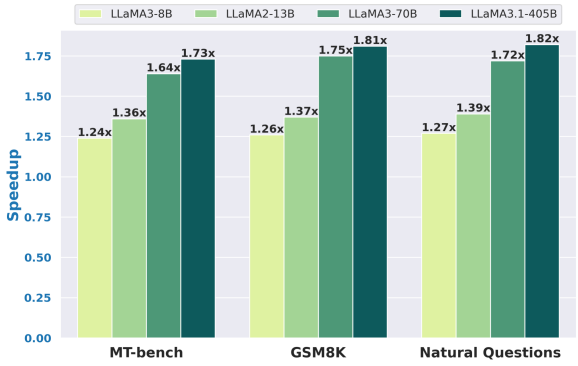


Figure 5: Model Size Scaling Laws of CLaSp.

plug-and-play solution, offering a versatile method to enhance inference speed for a range of LLMs.

5 Analysis

We present an extensive analysis of CLaSp, focusing on three key aspects: the benefits of the parallel strategy (Section 5.1), compatibility with different LLMs (Section 5.2), and the impact of key hyper-parameters (Section 5.3).

5.1 Sequence Parallel

As discussed in Section 3.6, our dynamic programming (DP) algorithm requires $\mathcal{O}(LM)$ layer forward passes. To assess the time overhead, we conducted experiments on LLaMA3-70B using two NVIDIA A800 GPUs. Without any parallel strategy, a single DP run to filter half of the layers takes approximately 2.5 seconds, whereas a single round of verification only takes about 0.1 seconds. After implementing our parallel strategy, the time for a single DP run is reduced to 0.14 seconds, comparable to the time for a single verification. This significantly reduces the additional latency introduced by layer optimization.

We further analyzed the per-query latency distribution of each stage, as illustrated in Figure 3c. The results show that the latency proportion of layer optimization is significantly reduced with the parallel strategy. Additionally, with a lower update frequency, the extra update latency of CLaSp becomes almost negligible.

5.2 Model Size Scaling Laws

To assess the scalability of CLaSp, we evaluated its performance across a range of model sizes, including LLaMA2-13B and LLaMA3.1-405B, in addition to LLaMA3-8B and LLaMA3-70B. For LLaMA2-13B, the model was deployed on an A800 GPU using FP16 precision, while for LLaMA3.1-405B, we used INT8 quantization (Dettmers et al., 2022) to deploy it on a single node with 8 A800 GPUs.

As illustrated in Figure 5, the speedup increases with model size across various tasks. Specifically, on the MT-bench, speedups range from $1.24\times$ for LLaMA3-8B to $1.73\times$ for LLaMA3.1-405B. For the GSM8K benchmark, speedups increase from $1.26\times$ to $1.81\times$, while on the Natural Questions benchmark, speedups range from $1.27\times$ to $1.82\times$. These results indicate that larger models exhibit enhanced layer sparsity, enabling CLaSp to leverage its capabilities more effectively and achieve greater speedups.

5.3 Key Hyper-Parameters

We show the effect of key hyper-parameters on the acceleration benefits of CLaSp, where all experiments were performed using the LLaMA3-70B model on MT-Bench.

5.3.1 Number of Skipped Layers

Layer sparsity allows intermediate layers to be skipped, but the number of skipped layers directly influences performance. Adjusting this parameter involves a trade-off between draft quality and efficiency, both of which significantly impact the speedup.

As shown in Figure 4a, for LLaMA3-70B, which consists of 80 layers, the speedup increases as the number of skipped layers rises, reaching an optimal value of $1.64\times$ when 44 layers are skipped. Beyond this point, the benefits of a longer average acceptance length are outweighed by the increased cost of generating high-quality drafts, resulting in a decline in speedup.

5.3.2 Layer Optimization Interval

Performing layer optimization after every verification step is computationally expensive, as noted in Section 3.7. Extending the Layer Optimization Interval (LOI) reduces the additional delays introduced by dynamic programming (DP) while having only a minor impact on the average acceptance length τ.

As illustrated in Figure 4b, the speedup initially increases with the LOI but begins to decline as the interval grows beyond 128. This decline is caused by a significant drop in τ, which negatively impacts overall speedup.

5.3.3 Draft-Exiting Threshold

To balance draft efficiency and cost, skipping 40% to 60% of layers achieves an optimal trade-off, as noted in Section 5.3.1. However, the cost of a single draft remains high, necessitating a sufficiently high acceptance rate to maximize speedup.

EAGLE-2 (Li et al., 2024a) suggests leveraging the draft model's confidence score to approximate the acceptance rate. By tuning the Draft-Exiting Threshold (DET), we can control the acceptance rate to optimize acceleration.

As shown in Figure 4c, adjusting the DET around 0.7 results in the highest speedup. Even with higher DET values, high speedup is maintained, demonstrating the robustness of this parameter for achieving acceleration gains.

6 Conclusion

In this paper, we propose **CLaSp**, a novel self-speculative decoding framework that adaptively adjusts the layer-skipping strategy based on context. We discover the potential of context-aware layer sparsity for generating high-quality drafts. Leveraging this insight, CLaSp performs layer optimization before each draft stage with minimal additional latency, significantly increasing the decoding efficiency. Through extensive experiments across diverse text generation tasks, we demonstrated that CLaSp achieves consistent speedups of $\mathbf{1.3}\times \sim \mathbf{1.7}\times$ over vanilla autoregressive decoding. Furthermore, detailed analysis reveals that CLaSp generalizes well to different models and tasks. For future work, we aim to explore ways to better leverage the layer sparsity of LLMs to further reduce inference latency in larger models.

Limitations

The CLaSp framework dynamically adjusts the layer-skipping strategy based on context, making the self-speculative decoding process of LLMs more efficient. However, certain limitations exist. Our experiments are conducted solely on NVIDIA A800 GPUs with 80GB of memory and limited to LLaMA series models, leaving the potential of more powerful hardware and other models unexplored. Additionally, while CLaSp can function alongside many existing speculative decoding innovations, we do not investigate these integrations. We believe that addressing these limitations and exploring such combinations in future research could lead to significant advancements.

Acknowledgments

Min Yang was supported by National Key Research and Development Program of China (2022YFF0902100), National Natural Science Foundation of China (Grant No. 62376262), the Natural Science Foundation of Guangdong Province of China (2024A1515030166, 2025B1515020032), Shenzhen Science and Technology Innovation Program (KQTD20190929172835662).

References

2024. Lisa: Layerwise importance sampling for memory-efficient large language model fine-tuning. *arXiv preprint arXiv:2403.17919*.

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

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. 2023. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

F. Warren Burton. 1985. Speculative computation, parallelism, and functional programming. *IEEE Transactions on Computers*, C-34(12):1190–1193.

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 *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

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

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. 2024. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *arXiv preprint arXiv:2402.12374*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Luciano Del Corro, Allie Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Awadallah, and Subhabrata Mukherjee. 2023. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference. *arXiv preprint arXiv:2307.02628*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*, volume 35, pages 30318–30332. Curran Associates, Inc.

Cunxiao Du, Jing Jiang, Yuanchen Xu, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, and Yang You. 2024. Glide with a cape: A low-hassle method to accelerate speculative decoding. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. 2024. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12622–12642, Bangkok, Thailand. Association for Computational Linguistics.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Theodore Glavas, Joud Chataoui, Florence Regol, Wassim Jabbour, Antonios Valkanas, Boris N. Oreshkin, and Mark Coates. 2024. Dynamic layer selection in decoder-only transformers. *arXiv preprint arXiv:2410.20022*.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. 2024. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1582–1595, Mexico City, Mexico. Association for Computational Linguistics.

John L. Hennessy and David A. Patterson. 2012. *Computer Architecture: A Quantitative Approach*, 5 edition. Morgan Kaufmann, Amsterdam.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. EAGLE-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. EAGLE: speculative sampling requires rethinking feature uncertainty. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. 2024. Kangaroo: Lossless self-speculative decoding via double early exiting. *arXiv preprint arXiv:2404.18911*.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and Beidi Chen. 2023. Deja vu: Contextual sparsity for efficient LLMs at inference time. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 22137–22176. PMLR.

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 *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, page 932–949, New York, NY, USA. Association for Computing Machinery.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

David A. Patterson. 2004. Latency lags bandwith. *Commun. ACM*, 47(10):71–75.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Apoorv Saxena. 2023. Prompt lookup decoding.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. 2024. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*.

Ruslan Svirschevski, Avner May, Zhuoming Chen, Beidi Chen, Zhihao Jia, and Max Ryabinin. 2024. Specexec: Massively parallel speculative decoding for interactive llm inference on consumer devices. *arXiv preprint arXiv:2406.02532*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3909–3925, Singapore. Association for Computational Linguistics.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. 2024a. Swift: On-the-fly self-speculative decoding for llm inference acceleration. *arXiv preprint arXiv:2410.06916*.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024b. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 7655–7671, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11263–11282, Bangkok, Thailand. Association for Computational Linguistics.

Kaiyan Zhang, Ning Ding, Biqing Qi, Xuekai Zhu, Xinwei Long, and Bowen Zhou. 2023. CRaSh: Clustering, removing, and sharing enhance fine-tuning without full large language model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9612–9637, Singapore. Association for Computational Linguistics.

Minjia Zhang and Yuxiong He. 2020. Accelerating training of transformer-based language models with progressive layer dropping. In *Advances in Neural Information Processing Systems*, volume 33, pages 14011–14023. Curran Associates, Inc.

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 *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.