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

Anonymous ACL submission

Abstract

Speculative decoding (SD) is a promising method for accelerating Large Language Model (LLM) decoding. The speedup efficiency of SD mainly depends on the consistency between the draft model and the verify model. However, previous drafting methods usually require to train extra modules, which are challenging to obtain and be consistent with different LLMs. In this paper, we introduce **CLaSp**, an in-context layer skip strategy for self-speculative decoding. It requires neither additional drafting modules nor additional training. Instead, it employs a plug-and-play method by skipping the intermediate layers of the verify model to be a compressed draft model. Specifically, we design a dynamic programming algorithm to skip layers for current drafting, which utilizes the full hidden states from last verify stage as optimization objective. Therefore, **CLaSp** can dynamically adjust the layer skipping strategy based on context after each verify stage, without pre-optimizing a fixed set of skipped layers on amounts of training data. Experimental results across various downstream tasks indicate that **CLaSp** achieved $1.3\times \sim 1.7\times$ speedup on LLaMA3 series models without altering the original distribution of the generated text.

1 Introduction

Transformer Large Language Models (LLMs) have achieved remarkable success in a wide range of natural language processing applications (Brown et al., 2020; Achiam et al., 2023). Scaling the model size and context window brings superior performance (Kaplan et al., 2020; Anil et al., 2023; Reid et al., 2024), but also leads to a rapid increase in inference latency. The inference latency is mainly attributed to the autoregressive nature of LLMs, where the model parameters will be loaded into the GPU SRAM for each token generation, resulting in underutilization of the computing cores during the decoding stage (Patterson, 2004; Shazeer, 2019; Agrawal et al., 2023).

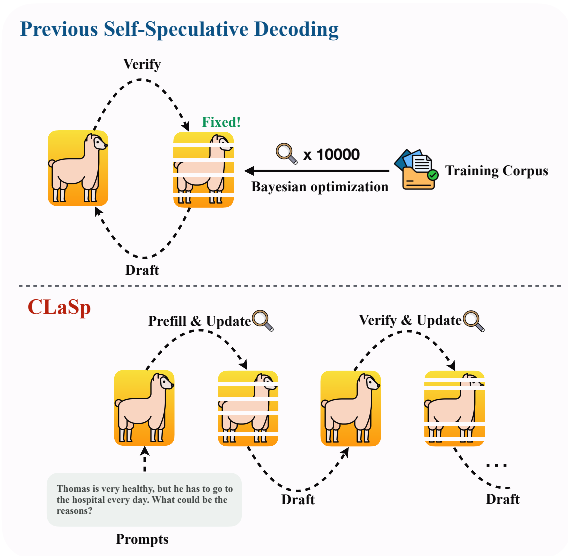


Figure 1: Previous Self-Speculative Decoding vs. **CLaSp**. Compared to the previous Self-SD method, which requires costly Bayesian optimization on training data to select a ***fixed*** set of skipped layers, CLaSp employs a ***dynamic*** skip-layer strategy that adjusts in real-time based on context.

Inspired by speculative execution (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. These methods employ an efficient draft model to quickly generate some draft tokens. Then, a slower LLM (referred to as the verify model) validates generated tokens in parallel by a single forward pass. Consequently, SD could effectively reduce the number of verify model's forward passes, alleviating the memory-bound problem caused by reading/writing of LLM parameters frequently.

The original SD requires to identify or train a suitable draft model that can generate outputs consistent with the verify model. This is friendly for some series of models that have already been open-sourced in different sizes (Touvron et al., 2023a,b; Dubey et al., 2024; Yang et al., 2024), but it's difficult to obtain a matching draft model for fine-tuned specialized models. To address this limitation, some previous method (Cai et al., 2024; Li et al., 2024b; Du et al., 2024; Liu et al., 2024) introduced additional lightweight modules as draft model to avoid retraining from scratch. However, they cannot generalize well to all tasks based on context, leading to a sharp drop in acceptance rate for some unseen tasks.

In parallel to introducing lightweight modules, Zhang et al. (2024) present a novel inference scheme, self-speculative decoding (Self-SD). Self-SD directly utilizes parts of the verify model as a compact draft model without any additional training. Specifically, it applies sparsification at the layer-level, skipping some intermediate layers of the verify model to create the draft model. Similar to methods that require training, it also lacks robust generalization and severely relies on an time-consuming Bayesian optimization process. SWIFT (Xia et al., 2024a) enhances Self-SD by dynamically optimizing the skipped layers as the number of user requests for the same task increases. However, when handling a single or a small amount of task data, SWIFT also exhibits poor performance.

Inspired by the contextual sparsity found in Deja Vu (Liu et al., 2023), we propose a dynamic in-context layer skip method (called **CLaSp**). Specifically, based on the observation of slowly changing embeddings across layers, we designed a dynamic programming algorithm to select the optimal skipped layer set with minimal additional latency. As shown in Figure 1, with its lower layer optimization latency, **CLaSp** can update the skipped layer set before each drafting step. It predicts the sparsity of the draft model ahead of the next drafting, leveraging the full hidden states from the last verification step as ground truth. Therefore, **CLaSp** identifies the most suitable draft model at each decoding step, maximizing the acceptance rate and thereby optimizing acceleration benefits. We conduct experiments using LLaMA3 series models on Spec-Bench (Xia et al., 2024b), a comprehensive benchmark designed for assessing speculative decoding methods across diverse scenarios. **CLaSp** achieves a $1.3\times \sim 1.7\times$ wallclock time speedup compared to conventional autoregressive decoding.

Our main contributions are summarized as follows:

- We introduce **CLaSp**, a self-speculative decoding framework that adaptively adjusts the layer skip strategy based on context.
- We propose additional performance optimization strategies in **CLaSp** to fully leverage GPU parallelism, making the extra latency from layer optimization almost negligible.
- We conduct comprehensive experiments on Spec-Bench, demonstrating that **CLaSp** consistently achieved $1.3\times \sim 1.7\times$ speedup without training. Additionally, a detailed analysis of key hyper-parameters further demonstrated the effectiveness of our method.

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 adding an additional lightweight module as a draft model. Medusa (Cai et al., 2024) additionally trained multiple decoding heads to predict the next n tokens in parallel. EAGLE/EAGLE-2 (Li et al., 2024b,a) adds only a lightweight plug-in (a single transformer decoder layer) to existing LLM. GliDe (Du et al., 2024) reuses the verify model's KV cache, and the proposed chunked attention mask method addresses the issue of context misalignment when using information from the verify model. However, they do not generalize well to some unseen tasks, resulting in only minor acceleration effects. REST (He et al., 2024) and Prompt Lookup Decoding (Saxena, 2023) replace specific draft model with retrieval, pulling relevant drafts from a text corpus or context based on input prompts. But this approach is highly task-sensitive and may not be suitable for all scenarios. Self-SD (Zhang et al., 2024) and SWIFT (Xia et al., 2024a) rapidly generates drafts by skipping intermediate layers of the original LLM without requiring additional draft models or modules. Triforce (Sun et al., 2024) using partial KV cache as draft model, full KV cache as verify model. In long context tasks, reducing the I/O operations of the KV cache can effectively decrease inference latency, as its memory footprint far exceeds that of the model weights. Jacobi Decoding (Santilli et al., 2023) and Lookahead (Fu et al., 2024) reformulates autoregressive decoding as a fixed-point Jacobi iteration, enabling the parallel generation of multiple tokens at each Jacobi decoding step. Although the above methods require no additional parameters and use part of the original LLM as a draft model, they lack the flexibility to dynamically adjust based on context, thus not maximizing the potential of the draft model. To enhance the acceleration effect of speculative decoding, tree attention (Miao et al., 2024; Cai et al., 2024; Chen et al., 2024; Svirschevski et al., 2024) has become an indispensable component. It extends from a single candidate sequence to a candidate tree, providing the verify model with more options.

Layer-wise Sparsity. Many previous studies have identified layer redundancy in LLMs as evidenced by 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). This suggests that the importance of each layer may vary, and not all layers are necessary. However, selecting the appropriate layers for different downstream tasks remains a significant challenge. Deja Vu (Liu et al., 2023) identifies the presence of context sparsity and leverages it to accelerate LLM inference without affecting the model's capabilities. LISA (pan, 2024) randomly selects a subset of layers to optimize during training, aiming for faster convergence and improved performance. Although these methods are effective, sparsification is lossy and cannot guarantee that the sparse distribution will perfectly match the original distribution. Glavas et al. (2024) discuss two common dynamic inference methods for natural language generation: layer skipping and early exiting. Unlike these previous methods, we focus on the layer-wise sparse strategy compatible with speculative decoding, enabling lossless inference acceleration.

3 CLaSp

In this section, we first introduced the complete pipeline of **CLaSp** from a global perspective. Then, we explore the main challenges (§3.2) faced by **CLaSp** and the problem formulation of layer skip (§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).

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 $\mathcal{S}$

3.1 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 from the drafting stage. This verification is completed 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. Once a draft token x_j is rejected, we use the original LLM's prediction to overwrite x_j and resume from token x_{j+1} for the next round of drafting. (3) layer optimization: using the hidden states of generating the last accepted token x_j as optimization objective, we update the optimal skipped layer set $\mathcal{S}^*$ to guide the next round of drafting process. In this way, before each round of drafting, we could update the draft model to better adapt to the current context.

3.2 Main Challenges

Compared to previous methods, **CLaSp** requires updating the skipped layer set before each draft step, which necessitates considering two main challenges: **1) How to determine which layers should be skipped?** This is the most critical issue that **CLaSp** aims to address, as it essentially determines the drafting quality. An ideal layer skip strategy

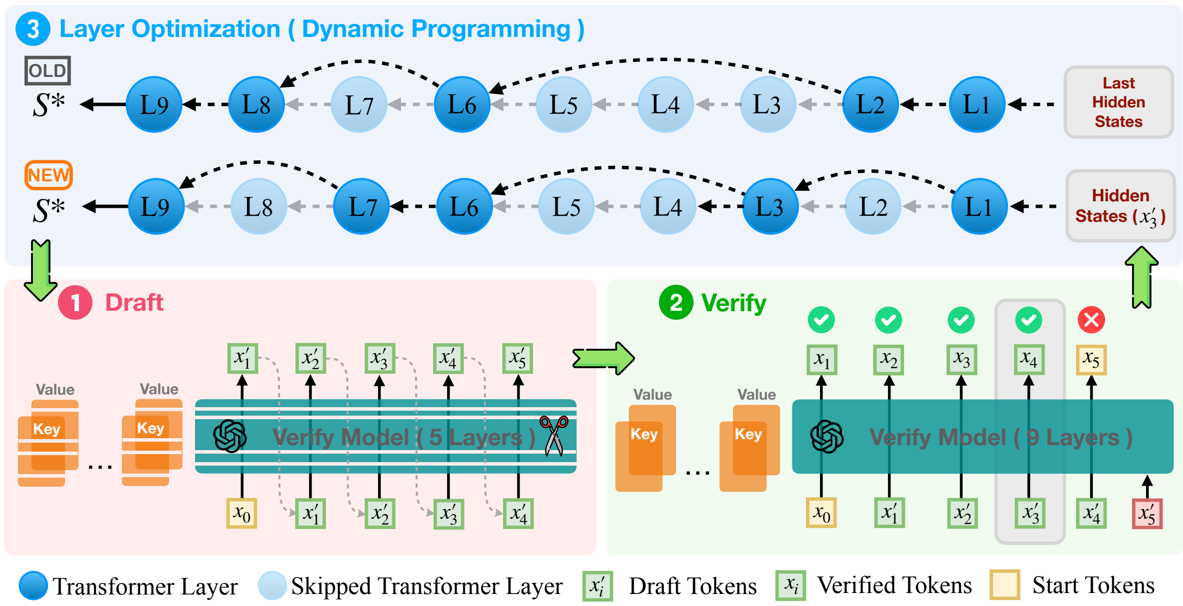


Figure 2: The overall framework of **CLaSp** consists of three stages: (1) Draft, (2) Verify, (3) Layer Optimization. After the Verify stage, **CLaSp** uses the information obtained to perform Layer Optimization, resulting in a new optimal layer skipping set $\mathcal{S}^*$. This set guides the next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

depends on the most recent context, ensuring that drafted tokens could be more likely to be accepted by the verify model. **2) How to reduce the additional latency caused by layer optimization?** Dynamic skipping strategy inevitably introduces additional computational delays, due to the multiple searches for the current optimal layer subset.

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 for selecting information for layer optimization is to avoid introducing additional computations, using information obtained from previous steps to reduce extra delays. We observed that after each verification step in speculative decoding, all the hidden states of the last accepted token are not fully utilized. So we aim to use this feedback information to predict the draft model for the next draft stage. Specifically, we denote the input tokens to a Transformer model as X, with an embedding layer that maps the token indices to token embeddings h_0. The transformer model has L transformer layers, where the l-th transformer layer evolves the transformation f_l: $h_{l+1} = f_l(h_l)$.

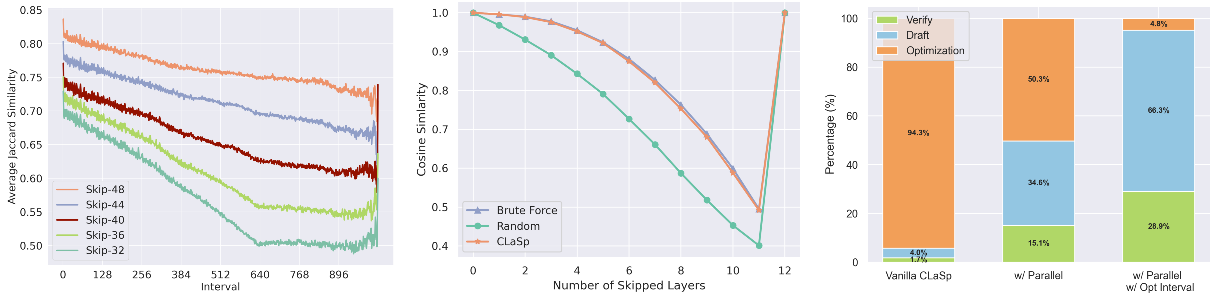
Let $\mathcal{D}(i,j)$ represent the maximum cosine similarity between h_i and the optimal hidden state $g(i,j)$ obtained by skipping j layers among the first i transformer layers. So we design a dynamic programming transition equation defined as:

$$\begin{aligned}\mathcal{D}(i,j) = \max\{&cosine(h_i, g(i-1,j-1)),\\ &cosine(h_i, f_{i-1}(g(i-1,j)))\}\end{aligned} \tag{2}$$

where $cosine$ is used to calculate the cosine similarity between two vectors. The **CLaSp** skip layer algorithm process is shown in Algorithm 1. A complete **CLaSp** process is elaborated in Figure 2.

3.5 Approximate Markov Property

A crucial prerequisite for dynamic programming algorithms is the 'no aftereffect' property, meaning that current decisions and state transitions are independent of previous states. However, when computing the optimal hidden states $g(i,j)$, **CLaSp**



(a) (b) (c)

Figure 3: (a) Observation of Sparse Persistence: the skipped layer sets selected for adjacent tokens have high similarity, and this similarity gradually decreases as the gap increases. Therefore, layer optimization can be performed on the current token to guide the subsequent draft process. (b) Approximate Markov Property: comparing the cosine similarity of hidden states obtained from Brute Force, Random, and **CLaSp**'s dynamic programming settings with the full forward pass demonstrates the approximate Markov property of **CLaSp**. (c) Efficiency Optimization Strategies: the latency breakdown per query indicates that the additional delay introduced by Layer Optimization accounts for only 4.8% of the total latency.

clearly does not have the Markov property, making it impossible to find an exact optimal solution using the Algorithm 1. Fortunately, due to the favorable property of slowly changing embeddings across layers, we find that **CLaSp**'s approximate algorithm is very similar to the brute force selected skipped layer set. we fix the first and last 10 layers of the 32-layer LLaMA3-8B model. Then, We compare the outcomes of a brute force search for the optimal solution, random layer selection, and **CLaSp** across the remaining 12 layers. As shown in Figure 3b, we find that the hidden states obtained by skipping the layers selected by **CLaSp** exhibit remarkable consistency with those obtained through brute force search. Both demonstrate a high cosine similarity with the hidden states from the original LLM. In contrast, the results from skipping randomly selected layers are relatively poor. Therefore, we can assume that **CLaSp** has the approximate markov property, finding the optimal solution 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 must be efficient enough to avoid additional delays while ensuring accurate drafting in subsequent decoding steps. Specifically, we use some parallel strategies to reduce 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 $\mathcal{O}(LM)$. When computing the state at (i, j), only the state at $(i-1, \cdot)$ is needed. Therefore, the computations for different j values with the same i are independent, allowing us to parallelize the second loop.

To reduce GPU memory footprint, we do not simply concatenate these states into a batch. Instead, we designed a special mask matrix that allows these states to parallelize like a sequence, reusing the same KV Cache without needing to duplicate it multiple times.

3.7 Lower Optimization Frequency

CLaSp needs to update the optimal skipped layer set after each verification based on the last accepted token, but the time cost of updating once is nearly the same as performing a verification, which becomes a bottleneck for the inference latency of **CLaSp**. Fortunately, we observe the phenomenon of Sparse Persistence: The set of skipped layers needed by adjacent tokens tends to be similar, so we calculate the Jaccard similarity between the sets of layers selected for skipping by adjacent tokens. As shown in Figure 3a, it can be observed that the similarity of the selected layer skipping sets only significantly decreases when the distance between two tokens exceeds a certain range. Based on the observation of Sparse Persistence, we further found that the optimal skipped layer set does not change drastically after each update. Therefore, we adjusted the update frequency, opting to update after accumulating several verification steps rather than after every single verification step. After adopting a lower update frequency, although the average acceptance rate of draft tokens decreased slightly,

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.

the overall benefits from reduced update latency resulted in a significant increase in the speedup ratio.

4 Experiments

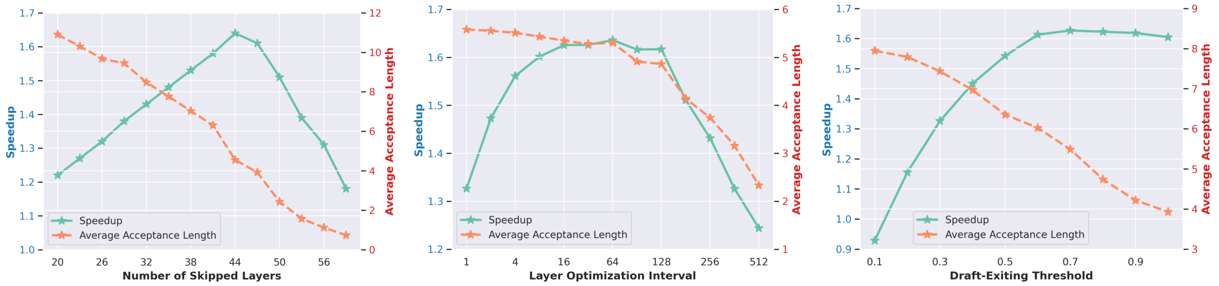
This section focuses on evaluating **CLaSp** on various text generation tasks to demonstrate the efficiency and effectiveness of **CLaSp**.

Model and testbed. We use four different sizes of LLaMA models (Dubey et al., 2024), including LLaMA3-8B, LLaMA2-13B, LLaMA3-70B and LLaMA3.1-405B, on NVIDIA A800 GPUs with 80GB of memory. The 8B and 13B model is deployed on a single A800, while the 70B and 405B models utilize 2 and 8 A800 GPUs respectively, with pipeline parallelism supported by Accelerate (Gugger et al., 2022). All models use FP16 precision except for LLaMA3.1-405B, which uses INT8 quantization. And for all models, if not specified, the batch size is 1.

Datasets. We benchmarked the performance of **CLaSp** on Spec-Bench (Xia et al., 2024b), which includes a wide range of datasets and tasks, covering six subtasks: multi-turn conversation, translation, summarization, question answering, mathematical reasoning, 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 generation length in above tasks, we set the maximum sequence length to 1024, aligned with prior setups (Xia et al., 2024b).

Comparison. In our main experiments, we use vanilla autoregressive decoding as the baseline, which serves as the benchmark for speedup ratios (1.00x). We compare **CLaSp** to existing training-free layer skip methods: Self-Speculative Decoding (Zhang et al., 2024) and SWIFT (Xia et al., 2024a). We exclude other SD methods from our comparison as they necessitate additional modules or extensive training, which limits their generalizability. The speedup ratio is hardware-dependent, so we tested different methods on the same devices to ensure fairness.

Performance Metrics. **CLaSp** is essentially still speculative sampling, which has been proven to be a lossless acceleration method (Leviathan et al., 2023). Therefore, we do not evaluate the generation quality and instead use the following metrics to assess acceleration performance: **Speedup Ratio**: The actual test speedup ratio relative to vanilla



(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.

autoregressive decoding. **Average Acceptance Length** τ: The average number of tokens generated per drafting-verification cycle, corresponding to the number of tokens accepted from the draft. The advantage of average acceptance length is that it is independent of hardware and runtime environment, while its disadvantage is that it does not reflect the overhead of the draft model.

4.1 Experimental Result

As shown in Table 1, we report the performance of **CLaSp** and previous plug-and-play methods on text generation tasks from Spec-Bench under greedy (Temperature=0) and non-greedy (Temperature=1) settings. The experimental results reveal the following findings: (1) **CLaSp** shows superior efficiency over prior methods, achieving consistent speedups of $1.3\times \sim 1.7\times$ over vanilla autoregressive decoding across various models and tasks. Prior methods rely on Bayesian optimization, exhibiting lower performance when the data volume is limited. (2) **CLaSp** consistently demonstrates significant improvements across average acceptance length, acceptance rate and speedups. This efficiency is primarily due to **CLaSp**'s ability to utilize the model's layer sparsity effectively. By skipping 50% to 60% of the layers during the experiments, **CLaSp** still maintains both a high average acceptance length and acceptance rate, which contributes to its superior acceleration. In most experimental settings, greater acceptance lengths generally lead to higher speedups. However, there are instances where the speedup ratio remains low despite having a long average acceptance length. This occurs because more time is spent drafting additional tokens, resulting in a lower acceptance rate and thus reducing the speedups. (3) In particular, the performance advantage of **CLaSp** is more pronounced on the LLaMA3-70B compared to LLaMA2-13B and LLaMA3-8B, which indicates that **CLaSp** can better leverage the greater layer sparsity present in larger models, enhancing its adaptability and efficiency.

Overall, the robust performance of **CLaSp** across different models highlights its effectiveness as a plug-and-play solution, offering a versatile method to enhance inference speed in a range of LLMs.

5 Analysis

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

5.1 Sequence Parallel

As mentioned in Section 3.6, our dynamic programming (DP) algorithm requires $\mathcal{O}(LM)$ layer forward passes. We conduct experimental analysis on the LLaMA3-70B using two NVIDIA A800 GPUs (80GB memory) to assess the actual time overhead. Without any parallel strategy, a single DP run to filter half of the layers takes about 2.5 seconds, whereas a single round of verification takes only about 0.1 seconds. After implementing our parallel strategy, the time for a single DP is reduced to 0.14 seconds, approximately equal to the time for a single verification, significantly reducing the introduced additional latency. We perform per-query experiments to analyze the latency distribution of each stage, as illustrated in Figure 3c. The latency proportion of layer optimization is significantly reduced with the parallel strategies. Additionally, with a lower update frequency, the extra update latency of **CLaSp** is almost negligible.

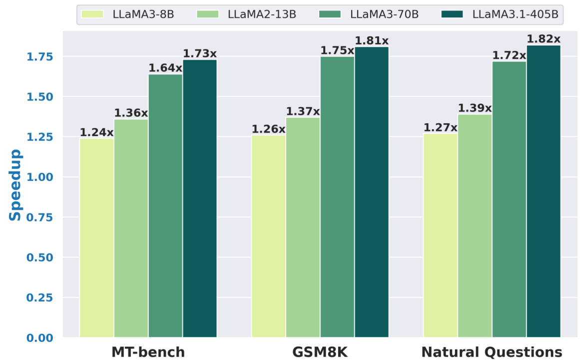


Figure 5: Model Size Scaling Laws of **CLaSp**.

5.2 Model Size Scaling Laws

Beyond LLaMA3-8B and LLaMA3-70B, we also assess the performance of **CLaSp** on other model sizes for text generation tasks, including LLaMA2-13B and LLaMA3.1-405B, to obeserve the impact of model size on acceleration performance. For LLaMA2-13B, we deploy it on a A800 GPU using float16 precision. While for LLaMA3.1-405B, we use 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 1.24x for LLaMA3-8B to 1.73x for LLaMA3.1-405B. For the GSM8K benchmark, speedups increase from 1.26x to 1.81x, while on the Natural Questions benchmark, speedups increase from 1.27x to 1.82x. These results indicate that larger models exhibit enhanced layer sparsity, allowing **CLaSp** to leverage its potential more effectively and achieve greater speedup.

5.3 Key Hyper-Parameters

5.3.1 Number of Skipped Layers

Since we utilize layer sparsity to skip the intermediate layers, it's important to assess how the number of skipped layers affects performance. Adjusting this number involves a trade-off between draft quality and draft efficiency, both of which significantly impact speedup. As shown in Figure 4a, for LLaMA3-70b which consists of 80 layers, the speedup increases with the number of skipped layers, reaching an optimal value of $1.64\times$ where number of skipped layers is 44. However, beyond this point, the advantages of a longer average acceptance length are offset by the increased cost of generating a high-quality draft, resulting in a decline in speedup.

5.3.2 Layer Optimization Interval

As mentioned in Section 3.7, performing layer optimization after each verification is prohibitively costly. By extending the update interval, additional delay introduced by DP can be significantly reduced with a minor impact on the average acceptance length. Figure 4b illustrates that the speedup initially rises and then falls as the Layer Optimization Interval (LOI) increases. Once the LOI surpasses 128, the substantial drop in τ leads to a notable decrease in speedup.

5.3.3 Draft-Exiting Threshold

As noted in Section 5.3.1, skipping 40% to 60% of layers achieves an optimal balance between draft efficiency and cost, resulting in an improved speedup. However, the cost of a single draft remains high, necessitating a sufficiently high acceptance rate for optimal speedup. Fortunately, EAGLE-2 (Li et al., 2024a) suggests using the draft model's confidence score to approximate the acceptance rate. By adjusting the Draft-Exiting Threshold (DET), we can control the acceptance rate to achieve optimal acceleration. Figure 4c shows the impact of the DET on speedup and the average acceptance length τ. The figure shows that setting the DET around 0.7 results in the highest speedup. Even as the DET increases, a high speedup can still be maintained.

6 Conclusion

In this paper, we propose **CLaSp**, a self-speculative decoding framework that adaptively adjusts the layer skip 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 speedup. We conduct extensive experiments across various tasks, demonstrating that **CLaSp** achieves over a $1.3\times \sim 1.7\times$ speedup. Furthermore, detailed analysis reveals that **CLaSp** generalizes well to different models and tasks. Additionally, an in-depth discussion of the hyper-parameters facilitates **CLaSp**'s adaptation to different LLM backbones. 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 skip 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.

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.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of LLM inference using lookahead decoding. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. 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*.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodola. 2023. Accelerating transformer inference for translation via parallel decoding. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12336–12355, Toronto, Canada. Association for Computational Linguistics.

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*.