S3D: A Simple and Cost-Effective Self-Speculative Decoding Scheme for Low-Memory GPUs

Anonymous ACL submission

Abstract

Speculative decoding (SD) has attracted a significant amount of research attention due to the substantial speedup it can achieve for LLM inference. However, speculative decoding methods often achieve optimal performance on high-end devices or with a substantial GPU memory overhead. Given limited memory and the necessity of quantization, a high-performing SD model on a high-end GPU can slow down by up to 7 times. To this end, we propose **S**kippy **S**imultaneous **S**peculative **D**ecoding (or S3D), a cost-effective self-speculative SD method based on simultaneous multi-token decoding and mid-layer skipping. When compared against recent effective open-source SD systems, our method has achieved one of the top performance-memory ratios while requiring minimal architecture changes and training data. Leveraging our memory efficiency, we created a smaller yet more effective SD model based on Phi-3. It is 1.4 to 2 times faster than the quantized EAGLE model and operates in half-precision while using less VRAM.

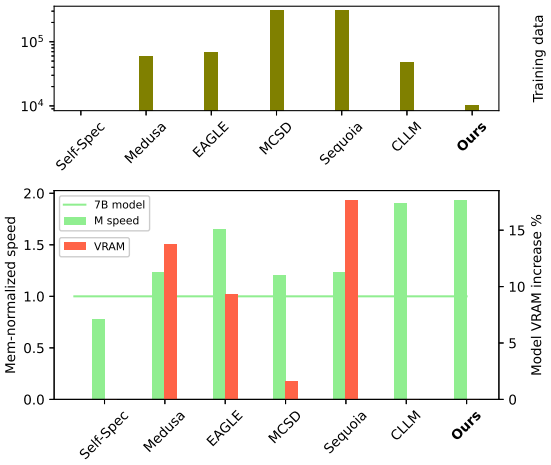

Figure 1: Training efficiency, inference efficiency per memory unit, and load-time VRAM evaluated for different models on MT-Bench. From left to right: The most recent open-source SD systems ordered by release dates. All systems use 7B target models with 8-bit quantization. Our model (S3D) stands out in both training efficiency and memory-speed trade-offs.

1 Introduction

Speculative decoding (SD) (Stern et al., 2018; Zhang et al., 2024; Xia et al., 2024) can accelerate LLM inference without sacrificing the quality. As a result, it is becoming one of the most common optimization techniques in LLMs. At a high level, typical speculative decoding (SD) works by *drafting* tokens at a relatively faster speed, and then *verifying* the guessed tokens at the end of an iteration using a full forward pass. The speedup is based on the assumption that the accepted tokens in one forward pass during the verification step will offset the cost of the drafting steps.

However, greater speedups are not always free. On one hand, some popular SD systems (Cai et al., 2024; Li et al., 2024; Chen et al., 2024) add a considerable amount of memory, e.g., due to the extra modules or a large token trees used for drafting. When models are deployed at scale, even a minor memory overhead can largely increase the cost of inference, given the high cost of using HBM VRAM in inference. On the other hand, high-performing SD can achieve remarkable speedups using a large model (Zhang et al., 2023a; Zhao et al., 2024; Yang et al., 2024) or on high-end GPUs (Zhang et al., 2023a; Chen et al., 2024; Kou et al., 2024; Elhoushi et al., 2024). However, we notice that these speedups become limited or even start underperforming when a smaller model or low-memory device is used where quantization is necessary. Surprisingly, the state-of-the-art open-source SD model (in speedups) may slow down by up to 7 times when applying quantization under constrained GPU memory, highlighting the significant overheads from quantization (Lin et al., 2024). In such cases, we question the cost-effectiveness

of existing SD methods, even if they show greater speedups on high-end GPUs.

Although recently Chen et al. (2024) designed a GPU-agnostic optimization by pre-profiling the GPU times for the draft and verify stages, their approach relies on the hard assumption of fixed acceptance rates among different levels of the draft token tree, making it less applicable to various SD methods. Additionally, the optimal trees have fewer differences for smaller models or slower GPUs, limiting their potential on low-end devices.

In this work, we introduce **S**kippy **S**imultaneous **S**peculative **D**ecoding (or S3D) to achive fast inference, low VRAM costs, and high training efficiency. Our key contributions are listed below:

Effective Self-speculative SD: We propose a simple and cost-effective self-speculative decoding scheme named *S3D* for low-memory GPUs. Our scheme features mid-layer skipping and simultaneous multi-token predictions, offering no added VRAM costs and high training efficiency. Compared to Zhang et al. (2023a), S3D overcomes the limited speedups in smaller models.

Optimal hyper-parameters: Instead of relying on statistical optimization, we formalize the relationship between the number of skipped layers and speedup in self-speculative decoding, as previously studied in Zhang et al. (2023a) empirically. Based on our formalization, we can also verify the optimal number of token predictors, as empirically observed by Gloeckle et al. (2024).

Optimal speed-memory ratio: Our SD method demonstrates optimal performance-memory ratios among recent open-source SD models. By exploiting the memory efficiency, we can avoid the significant quantization overheads under certain VRAM constraints and outperform the previous fastest SD method, i.e., EAGLE (Li et al., 2024), under 8-bit quantization by up to 3.9x in speedups on an A10G GPU. Moreover, by switching to a smaller target model, we have created a more effective SD model based on Phi-3, which decodes 1.4 to 2 times faster than EAGLE on an A10G while using less VRAM.

2 Related Work

Early work in speculative decoding (SD) using Transformers (Stern et al., 2018; Sun et al., 2021; Xia et al., 2023) focused on in-domain tasks such as translation and grammar error correction, where significant speedups are easily achieved. These methods are characterized by single-branch speculation using additional modules (Stern et al., 2018) or an independent draft model (Xia et al., 2023). By using speculative sampling (Chen et al., 2023), SD can also sample tokens from target model distribution (Leviathan et al., 2023). In contrast, our work targets general domain tasks and focuses on greedy[1] and non-batching decoding via *simultaneous multi-token prediction*. We save memory and gain training efficiency through *layer-skipping*.

2.1 Multi-Token Predictions

Since Stern et al. (2018), predicting the next k tokens simultaneously has proven effective, but it requires adding k feed-forward decoder layers from the last encoder state. Cai et al. (2024) popularized this idea using *Medusa heads*, additionally predicting multiple token branches using tree attention (Miao et al., 2024; Spector and Re, 2023).

In the SpecDec method (Xia et al., 2023), multi-token prediction is done by unmasking future tokens from multiple decoder heads attending to different encoder states, utilizing distinct attention queries for predicting different tokens. However, SpecDec requires full model fine-tuning for all layers as the decoder-only target model has not been pretrained on multi-token unmasking tasks. More recently, Bhendawade et al. (2024) predict multiple tokens by adding streaming embeddings initialized from upper layers, with the token tree reduced by early exiting.

Multi-token prediction can also be implemented auto-regressively (Yang et al., 2024; Li et al., 2024; Ankner et al., 2024), which takes multiple steps to predict the next draft token conditioned on previously drafted tokens in one iteration. To mitigate the substantial overheads incurred by multi-step drafting within a single iteration, the draft overhead should be minimal while ensuring it retains the capability to generate acceptable tokens. In the case of EAGLE (Li et al., 2024), this is achieved by efficiently utilizing the target model's high-level features with the embeddings or hidden states of the next tokens for regression via an additional layer of Transformer decoder.

Another line of work to generate multiple draft tokens in parallel is based on Jacobi iteration methods, treating auto-regressive decoding in LLM as a non-linear system of equations, or Jacobi decod-

[1] Our approach can be easily extended to support sampling; we focus on greedy decoding as it is orthogonal to speculative sampling.

ing (Song et al., 2020; Santilli et al., 2023). In practice, however, an LLM may obtain marginal speedups from Jacobi decoding as it can rarely produce an accepted token if a previous token in the trajectory is predicted incorrectly. Lookahead decoding (Fu et al., 2024) attempts to address this issue by introducing memory costs and caching n-gram tokens from previous Jacobi trajectories. Inspired by the Consistency Model (Song et al., 2023), CLLMs (Kou et al., 2024) additionally train their target models to minimize the distances between Jacobi trajectories and the fixed point, leading to faster convergence and thus greater speedups. Compared to regular SD methods, Jacobi decoding does not have a separate draft phase.

2.2 Layer Skipping

Layer skipping is a type of structured pruning technique (Anwar et al., 2017; Louizos et al., 2018; Xia et al., 2022) that reduces a model by only using a subset of its layers. Structured pruning is particularly intriguing for LLM optimizations due to its compatibility with GPU acceleration. This is because it enables immediate gains in memory and compute by discarding substructures entirely (Ouderaa et al., 2024).

Various layer skipping schemes explored for Transformer models are discussed next. Early Exiting (Dehghani et al., 2019; Teerapittayanon et al., 2016) utilizes early layers and skips the rest. LayerDrop (Fan et al., 2020) randomly drops layers during training and skips layers during inference. Lagunas et al. (2021); Ouderaa et al. (2024) have identified sub-layer structures (e.g., attention heads or weight blocks) to be pruned during training. Sajjad et al. (2023) propose symmetric dropping of complete top and bottom layers, based on the observation that middle layers of a Transformer are less critical. This observation has been validated by Ma et al. (2023); Wu and Tu (2024) on larger LLMs and for KV-cache as well. Recently, Raposo et al. (2024) have trained additional parameters to dynamically skip layers.

Combining layer skipping with SD leads to an intriguing way to save memory, known as *self-speculative decoding* (Bae et al., 2023; Zhang et al., 2023a; Liu et al., 2024a; Elhoushi et al., 2024), where a static or adaptive number of layers of the target model are used for drafting tokens. These approaches mitigate the common memory overheads of SD by incorporating minimal or no extra modules for the draft stage. Specifically, they either entail no additional training (Zhang et al., 2023a) or only necessitate training for learning an adaptive early exit threshold (Bae et al., 2023; Elhoushi et al., 2024) with a potential trade-off in quality. In Elhoushi et al. (2024), they reuse KV-cache from draft stages, reducing the computation needed for the remaining layers in the verify stage. However, their approach involves training LayerDrop (Fan et al., 2020) and requires complex early exit thresholds during inference.

A concurrent work by Gloeckle et al. (2024) combines self-speculative decoding with multi-token predictions. However, their approach trains additional independent heads comprising entire Transformer layers, potentially adding more memory cost compared to EAGLE (Li et al., 2024). In contrast, our SD scheme imposes no extra model load cost and has minimal training requirements through applying mid-layer skipping.

3 Preliminaries

Given a Transformer decoder-only model (Radford et al., 2018) M_p, its generated next-token distribution $p(t_{\leq i})$ given the current context tokens $t_{\leq i} = t_1, t_2, ..., t_i$ can be expressed in terms of Transformer layers. For layer $\ell = 1, 2, ..., L$,

$$h_i^{(0)} = \text{Emb}(t_i) \quad (1)$$

$$h_i^{(\ell)} = T^{(\ell)}(h_{\leq i}^{(\ell-1)}, \text{Pos}_{\leq i}) \quad (2)$$

$$t_{i+1} \sim p(t_{\leq i}) = \text{LM-Head}(h_i^{(L)}) \quad (3)$$

where Emb represents the embedding transformation, and $T^{(\ell)}$ denotes the Transformer layer at level l, which receives the context hidden states from the previous layer $h_{\leq i}^{(\ell-1)}$, associated with their position information $\text{Pos}_{\leq i}$. The LM-Head maps the hidden space to a vocabulary distribution p for the next token sampling.

The decoder-only language model is typically trained using the next-token prediction task, where training involves employing cross entropy loss across tokens in parallel. Given a sample of sequential tokens $t_i, i = 1, 2, ..., N$, the loss is

$$\mathcal{L} = \frac{1}{N-1} \sum_{i=1}^{N-1} -\log p(t_{\leq i})_{t_{i+1}} \quad (4)$$

During a SD iteration, a more efficient draft model M_q is often used to predict the next γ token(s) from the target model M_p through sampling $t_{i+j+1} \sim q(t_{\leq i+j})$ where $j = 0, 1, 2, ..., \gamma - 1$. To

produce tokens as if they were sampled from the target distribution, Leviathan et al. (2023) show that we can verify drafted tokens by comparing $p(t_{i+j})$ with $q(t_{i+j})$ successively, and accept each token with a probability of $\min(1, \frac{p(t_{i+j})}{q(t_{i+j})})$. Upon completion, one more last token $t_{i+\gamma+1}$ can be sampled from the target distribution p. On rejection, sampling is done from a normalized distribution of $\max(0, p - q)$. In greedy decoding, this process is equivalent to accepting only the matched tokens produced from p and q.

4 S3D

We propose a self-speculative SD scheme called **S**kippy **S**imultaneous **S**peculative **D**ecoding (or S3D). In S3D, the draft model M_q uses partial layers from target model M_p.

To adhere to the Transformer decoder architecture and circumvent the need for auxiliary modules, we opt to emulate the Masked Language Modeling (MLM) task commonly employed in Transformer encoder training. This involves inserting a special mask token, denoted as <M>, into future inputs to predict the next γ tokens concurrently. Specifically, draft model receives the last token, and $\gamma - 1$ mask tokens:

$$t_{i+1}, ..., t_{i+\gamma} \sim q(t_{\leq i}, \underbrace{\texttt{<M>}, ..., \texttt{<M>}}_{\gamma-1}) \quad (5)$$

where the draft model M_q uses all previous hidden states of the target model $h^{(\ell)}_{\leq i}, \ell = 1, 2, ..., L$:

$$h^{(\ell)}_i = T^{(\ell)}(h^{(\ell-1)}_{\leq i}, \text{Pos}_{\leq i}). \quad (6)$$

Different from Xia et al. (2023), the simultaneously generated tokens at $j = i+1, i+2, ..., i+\gamma$ require only propagating through lower and top layers, skipping middle m-th to n-th layers of the target model:

$$h^{(n-1)}_j = h^{(m)}_j \quad (7)$$

$$h^{(\ell')}_j = T^{(\ell')}(h^{(\ell'-1)}_{\leq j}, \text{Pos}_{\leq j}) \quad (8)$$

where non-skipping layers $\ell' = 1, 2, ..., m, n, n+1, ..., L$. Unlike Zhang et al. (2023a), a non-skipping layer ℓ is able to utilize previous states of the target model, i.e., $h^{(\ell-1)}_{\leq i}$. Furthermore, in contrast to early exiting in Zhang et al. (2023a), the current draft states from top layers are kept for decoding. Additionally, we do not necessarily skip lower layers due to the adjustments required in lower-level representations for skipped middle layers, a notion explored similarly by Ma et al. (2023) and Wu and Tu (2024). We will further justify this skipping scheme in Section 5.3.

Training: Our training objective is to accurately uncover masked tokens while preserving the original next-token prediction capability. To this end, we train the draft model to decode both the next token right after i and its following masked tokens. Assume the masked tokens are located at $i+1, i+2, ..., i+\gamma-1$, our training loss is

$$\mathcal{L}^{(S3D)} = \frac{1}{|D|} \sum_{j \in D} -\log q(t_{\leq j})_{t_{j+1}} \quad (9)$$

where decoding set $D = \{i, i+1, ..., i+\gamma-1\}$.

During training, we freeze the skipped layers to preserve the target model distribution. Instead of predicting next tokens sequentially, we assign masked tokens randomly so that training samples can be processed in one batch, utilizing the parallelism of Transformer. An illustration of our modeling is shown in Figure 2.

Predicting speedup: Given target ratio $\beta \in [0, 1]$, which represents the ratio of target model parameters that the draft model uses during decoding, the acceptance rate α of the first drafted token should be a function of β. Naturally, in the self-speculative case, $\alpha(1) = 1$ and $\alpha \to 0$ when $\beta \to 0$.

In this work, we have hypothesized a function to estimate draft token acceptance rate as a function of model size (parameterized by U):

$$\alpha(\beta; U) = \frac{1 - U^\beta}{1 - U}. \quad (10)$$

We will show in Section 5.3 that the above function aligns well with empirical observations.

In multi-token predictions, assume the true acceptance rate at the k-th draft token, i.e., $\alpha_k(\beta)$, is discounted by k in a discrete function (which we may readily estimate from empirical data). Following the notation in Li et al. (2024), the expected newly generated tokens τ is [2]

$$\tau(\gamma, \beta) = \sum_{n=1}^{\gamma+1} n \cdot \prod_{k=1}^{n-1} \alpha_k(\beta) \cdot z_n(\beta) \quad (11)$$

where the shorthand notation $z_n(\beta) = 1 - \alpha_n(\beta)$ if $n \neq \gamma + 1$ and 1 otherwise.

[2] When drafting a single next token, i.e., $\alpha_k(\beta) = \alpha_1(\beta)$, Eq. 11 is a capped geometric series and can be further simplified to a formula given by Leviathan et al. (2023).

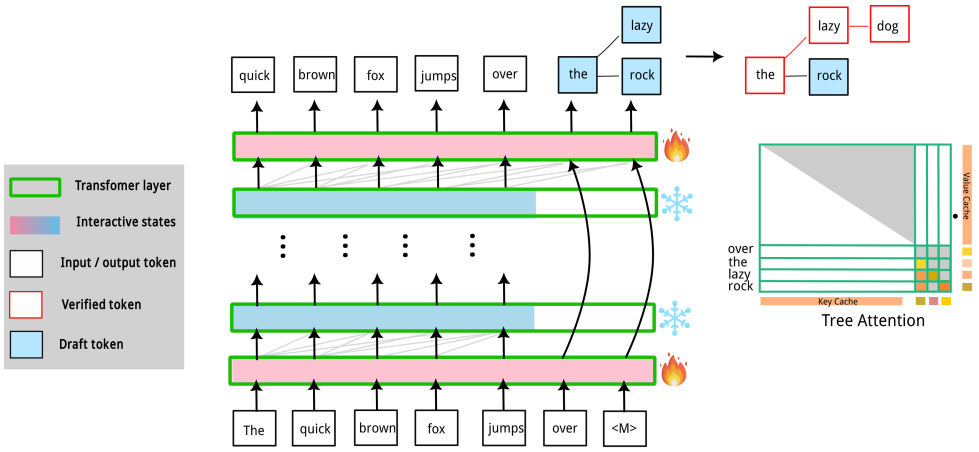

Figure 2: An illustration of S3D based on simultaneous predictions of the last γ tokens ($\gamma = 2$). A mask token <M> is added into vocabulary prior to training, and a partial model is trained to predict the next tokens simultaneously. Tree attention is adopted to verify multiple branches of predictions give top candidates of the k-th draft token. Unlike other self-speculative decoding methods based on fully-skipped layers, we only skip the middle layers on top of the draft tokens so that the draft model can access high-level features from top layers as well as the previous states verified by the complete target model.

When the number of drafting tokens is not significant (which commonly happens on low-end devices), it is reasonable to assume the time cost for a full-model forward pass to be a constant C. We also assume there is a fixed overhead H_0 for each iteration, proportionally to C, i.e., $H_0 = \delta \cdot C$, then deriving the decoding speed by taking out C leaves us the *Improvement Factor* (Leviathan et al., 2023) for self-speculative decoding, i.e.,

$$IF(\gamma, \beta) = \frac{\tau(\gamma, \beta)}{\delta + \beta + 1}, \tag{12}$$

assuming that the forward time for a partial model scales linearly with the number of its parameters.

Note that Eq. 12 represents a different improvement factor compared to the one in auto-regressive drafting schemes (Leviathan et al., 2023), where γ predictions are performed sequentially.

5 Experiments

5.1 Experimental Setup

Datasets We consider datasets commonly used in SD evaluations, including MT-Bench (Zheng et al., 2023) for multi-turn conversation, Human-Eval (Chen et al., 2021) for coding, and CNN-Daily (Hermann et al., 2015) for summarization. For CNN-Daily, we only use the $1,000$ samples similar to Zhang et al. (2023a), while the complete datasets are used for the others. For MT-Bench, we use Gemini Pro (Anil et al., 2024) for evaluation. We report accuracy scores for Human-Eval and Rouge-1 and Rouge-L scores (Lin, 2004) for CNN-Daily.

Baselines A fair comparison is conducted by running all systems on identical samples and hardware using a uniform evaluation framework. To this end, our model is compared to open-source SD systems including: Self-Spec (Zhang et al., 2023a), Medusa (Cai et al., 2024), EAGLE (Li et al., 2024), MCSD (Yang et al., 2024), Sequoia (Chen et al., 2024), Ouroboros (Zhao et al., 2024), and CLLMs (Kou et al., 2024). Self-Spec is a representative training-free self-speculative method that predicts the next single token via layer skipping. Medusa is a popular SD method that adds parallel decoder heads to predict multiple next tokens. EAGLE concatenates the target model's late-layer hidden states with the last token embedding to predict the next five tokens auto-regressively via an additional Transformer decoder layer. In a recent benchmark (Xia et al., 2024), it reportedly achieves the highest speedup. And recent work MCSD, Sequoia, and Ouroboros generate draft tokens through a separate draft model. In particular, Sequoia constructs an optimal draft token tree from profiling the underlying hardware. In these three systems, we adopt the 68M JackFram LLaMA (Miao et al., 2024) as the draft model, which is also the default and most efficient option for their LLaMA target models. Lastly, CLLMs is considered as the latest development in the direction of Jacobi or Lookahead decoding (Santilli et al., 2023; Fu et al., 2024).

Table 1: The cost-effectiveness comparisons on an A10G GPU considering peak VRAM costs. All models are 8-bit quantized and are based on the 7B LLaMA-v2 target model except mentioned otherwise in parentheses. The largest 3 numbers in each column are highlighted in *italics* or **bold**. "Peak" denotes the peak VRAM usage in GiB. The overall (averaged) results count for both M speed and the relative effectiveness metrics compared to the baseline.

Model \ Metric	MT-Bench				Human-Eval				CNN-Daily					Overall	
	Peak ↓	Tok / s	M	Score	Peak ↓	Tok / s	M	Acc. %	Peak ↓	Tok / s	M	R-1	R-L	M	Eff.
Baseline	*8.53*	7.02	1.00	7.05	*8.43*	7.02	1.00	6.71	*8.96*	6.53	1.00	0.19	0.13	1.00	1.00
Self-Spec.	7.77	5.00	0.78	**7.08**	7.46	4.89	0.79	6.10	8.09	5.01	0.85	0.19	**0.14**	0.81	0.97
Medusa (Vicuna)	*9.09*	9.27	1.24	4.98	*8.94*	10.69	1.44	7.93	*9.36*	7.62	1.12	**0.24**	**0.14**	1.26	1.03
EAGLE	*9.58*	**13.03**	1.65	6.98	*9.45*	**15.22**	1.93	5.49	*9.86*	**12.43**	**1.73**	0.19	0.13	1.77	0.94
MCSD	7.76	7.72	1.21	6.79	7.40	7.77	1.26	**9.76**	8.10	6.74	1.14	0.20	**0.14**	1.21	**1.16**
Sequoia	8.44	8.64	1.24	6.46	8.35	9.01	1.30	3.05	8.57	7.99	1.28	0.18	0.12	1.27	0.77
Ouroboros	7.95	5.47	0.84	**7.08**	7.61	5.83	0.92	**8.54**	8.30	4.91	0.81	0.18	0.13	0.86	**1.09**
CLLM	7.51	11.75	**1.90**	5.31	7.37	**16.29**	**2.66**	3.66	7.53	8.06	1.47	0.20	**0.14**	**2.01**	0.79
Ours															
S3D	7.79	**12.39**	**1.93**	5.68	7.60	13.85	**2.19**	6.71	8.80	**9.58**	**1.49**	**0.28**	**0.19**	**1.87**	**1.09**
S3D (Phi-3, fp16)	8.14	**25.31**	**3.78**	**7.04**	7.92	**28.13**	**4.27**	**20.12**	8.87	**17.35**	**2.69**	**0.25**	**0.18**	**3.58**	**1.77**

Implementation details All implementations use eager-mode decoding based on Huggingface Transformers (Wolf et al., 2020) and we adopt the native BNB 8-bit quantization (Dettmers et al., 2022) for quantized models.

We configure each system with greedy decoding and keep the other SD configurations default for different systems. A non-speculative implementation is used as the common baseline to calculate speedups and relative memory costs as it shares the same instructional LLaMA-v2 (Touvron et al., 2023) backbone for most of our evaluated systems.

We mostly consider the 7B target model size. In exception to this, we also train a 3.8B Phi-3 Mini (Abdin et al., 2024) target model and a 13B LLaMA-v2 target model to demonstrate the generalization ability of our method. Unless specified otherwise, our S3D implementations use the optimal hyper-parameters suggested by Section 5.3.

Cost effectiveness metric We propose a memory-normalized speed metric M, which divides a speedup by the relative memory usage compared to the baseline model:

$$M = \frac{v_1}{v_0} / \frac{m_1}{m_0} = \frac{v_1}{m_1} / \frac{v_0}{m_0} \tag{13}$$

where v_0 and m_0 are the generation speed and memory consumption of the baseline model, and v_1 and m_1 are the generation speed and memory consumption of the evaluating model. This metric quantifies the generation speedup per memory unit, ensuring a fair and memory-aware comparison for target models of the same size.

Training Similar to Medusa, EAGLE, CLLMs, et al.(Cai et al., 2024; Li et al., 2024; Kou et al., 2024), we train our models on the ShareGPT dataset. All training is conducted using bf16 and FlashAttention-2 (Dao, 2024) with a batch size of 64 for one epoch on A10G GPUs. Please refer to Appendix A for detailed training descriptions.

5.2 Main Results

Initially, we discuss the cost-effectiveness of our model. In Table 1 and 2, we observe that our S3D models and CLLM exhibit the highest overall M speeds among the evaluated systems, which remains consistent across different GPUs.

Importantly, models producing high speedups are not necessarily the most cost-effective, as seen in the case of Medusa and EAGLE, where the cost of extra draft module(s) must be considered. We also observed discrepancies in the effectiveness scores of models when using the original target model for token verification, likely due to implementation issues or numerical errors. However, our model generally maintains baseline effectiveness and achieves the highest overall effectiveness among systems requiring target model training (i.e., Self-Spec., CLLM, and ours). By using layer adapters like LoRA (Hu et al., 2022), we can easily enable lossless decoding at the expense of efficiency costs (see analysis in Appendix B).

Interestingly, the vanilla self-speculative decoding method, i.e., Self-Spec, underperforms the baseline in terms of speed for the 7B target model. This highlights the limitation of naive self-speculative decoding ($\gamma = 1$) in smaller models, where the partial draft model becomes further constrained and is unable to propose good draft tokens. In contrast, we alleviate this issue by allowing the draft model to attend to previous target model states and training the model to predict multiple tokens ($\gamma > 1$), thereby enhancing the effectiveness of smaller self-speculative models.

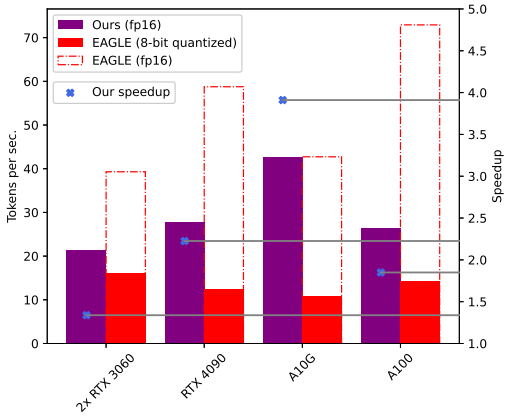

Figure 3: Speed comparison between ours (S3D) and EAGLE on different GPU devices (MT-Bench samples, 7B LLaMA target model). The dashed bars represents the full speed potentials of the EAGLE model without memory restrictions. However, when constrained with a VRAM limit of 16 GiB, the quantized EAGLE model (indicated by red bars) suffers from severe speed degradation, highlighting the significant overheads associated with quantization.

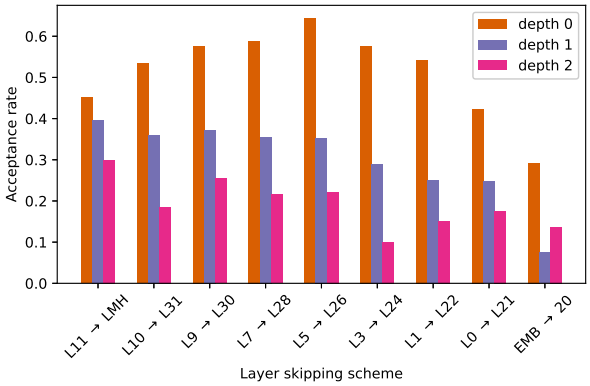

Figure 4: The overall acceptance rates and individual acceptance rates at different drafting depths (w/ only a single branch of future tokens). L, LMH, and EMB stand for regular layer, LM heads, and the embedding layer, respectively. Skipping the middle layers symmetrically has shown better acceptance rates in general. Note that we distinguish embedding layer and lm_head here although in practice they may have tied weights.

Admittedly, the LLaMA-based S3D model ties closely to EAGLE and underperforms CLLM in overall M speed, primarily due to the high speedups and optimal memory efficiency achieved by EAGLE and CLLM, respectively. However, as shown in Table 1 and 2, we are able to exploit our memory efficiency and outperform EAGLE in both efficiency and effectiveness while using less amount of VRAM by switching to a non-quantized Phi-3 target model. [3] Even without switching to a different target model, we demonstrate in Figure 3 that our LLaMA-based S3D model can operate in half-precision within a VRAM limit of 16 GiB, and outperform EAGLE by up to 3.9 times when EAGLE needs to be quantized. This underscores the critical importance of memory efficiency.

On the other hand, we find that the training objectives of CLLM may encourage repeating patterns in its outputs, leading to degraded effectiveness scores, as seen in Table 1 and additional case studies in Appendix D. In contrast, our model can preserve effectiveness scores more robustly while achieving the optimal speed-memory ratios.

5.3 Optimal Hyper-Parameters

We first study the optimal layer skipping choices. To this end, we have empirically explored three different schemes by skipping a fixed number of original layers in the LLaMA 7B target model: (1) Skipping asymmetrically from the middle, including early existing and using only late layers while skipping all early layers. (2) Skipping symmetric layers from the middle, i.e., layer 5 $\rightarrow$ 26 or skipping the middle 20 layers. (3) Alternate evenly between skipping and non-skipping layers.

Looking at Figure 4 (and Appendix Figure 9 for training efficiencies), skipping symmetrical layers from the middle performs best, achieving higher overall acceptance rates and optimal training efficiency. In contrast, skipping from layer 11 to the top (LM-head) layer and skipping from the bottom (embedding) layer to layer 20 have the worst performance, highlighting the importance of both early and late layers. Lastly, skipping symmetric middle layers or alternating every 3 layers has similar training efficiency.

To answer what is the optimal number of layers and what is the best number of tokens to be predicted in parallel, we train different number of layers (β) skipped symmetrically from the middle layer, each trained model is evaluated for different γ values up to 5. We run different models on MT-Bench for 50 samples, and linearly interpolate the acceptance rate discount function $\alpha_k(\beta)$ (detailed in Appendix E).

As summarized in Figure 5, our proposed formula for predicting self-speculative acceptance rates in Eq. 10 mostly matches with the empiri-

[3] We conducted ablations in Appendix C to understand the significant improvements in Phi-3 S3D. For a larger 13B model, our cost-effectiveness can be maintained as well (see Appendix Table 3).

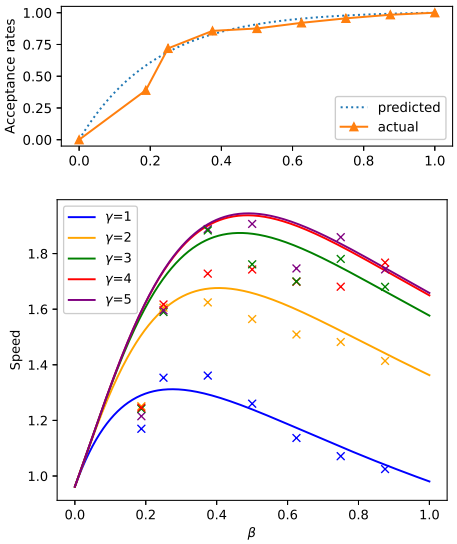

Figure 5: **Upper**: The predicted (in dashes) and sampled acceptance rates (interpolated orange dots) of various draft model sizes (β). **Lower**: The predicted (in curves) and sampled (in dots) speeds of different draft model sizes and different number of guesses (γ). All experiments are evaluated using MT-Bench. Our prediction curves justify the optimality of using around half the number of layers and $\gamma = 4$, as observed individually and respectively in Zhang et al. (2023a) and Gloeckle et al. (2024).

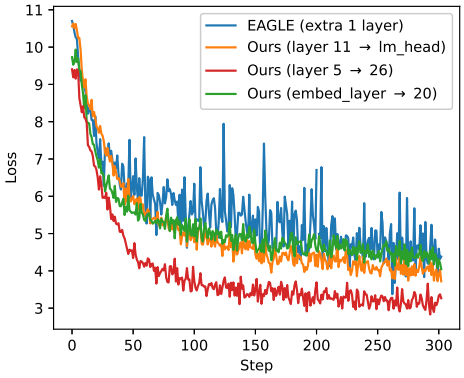

Figure 6: Training loss comparisons between EAGLE's classification loss (Li et al., 2024) and our (S3D) training loss in Eq. 9. EAGLE requires training an extra layer of Transformer with additional linear mappings.

cal results except for the lowest β value, and this outlier may be explained by the less predictability in training a small partial model (< 1.5B). Addtionally, the speedup formula in Eq. 12 successfully predicts both the trend and the sweet spot in speedups. Higher γ values align less with the prediction because the acceptance rates for far-future tokens have higher uncertainty and variance as reflected by Figure 4.

In addition to the findings from Zhang et al. (2023a); Gloeckle et al. (2024), we have unified multi-token predictions with layer skipping. Our prediction in Eq. 12 has also justified their findings that the optimal speed for single next-token prediction is achieved by skipping around half of the layers and the overall optimal γ is 4 (as shown in Figure 5, a higher γ results in an almost diminished speedup, offset by fewer accepted tokens).

5.4 Training Efficiency

In addition to its cost-effectiveness, S3D also demonstrates greater training efficiency compared to other effective SD models (see Figure 1). So we hypothesize that the self-speculative decoding method used in S3D inherently lowers training costs, as the training task leverages the existing model weights. In light of this, we train and compare both S3D and EAGLE models using $20,000$ data rows (the original EAGLE was trained on $68,000$ data rows).

As shown in Figure 6, S3D consistently demonstrates lower training losses, even when accounting for more layers and the inclusion of far-future tokens, which are typically difficult to predict. Remarkably, even in the least favorable scenario involving early exiting or early layer skipping, our loss values generally remain lower than those of EAGLE. Overall, S3D training exhibits less variance and achieves more stable convergence.

6 Conclusion

We have proposed S3D, a self-speculative SD method based on simultaneous multi-token predictions and mid-layer skipping. S3D demonstrates one of the best cost-effectiveness among recent open SD systems, while also exhibiting high training efficiency and maintaining the effectiveness of the original model. We have also verified the optimal hyper-parameters for our proposed method in a principled manner, without requiring any black-box optimizations beforehand. By leveraging memory efficiency, S3D can avoid quantization and surpass the speed of quantized EAGLE when a 16 GiB VRAM limit is imposed. Additionally, S3D, based on the smaller Phi-3 target model, decodes 1.4 to 2 times faster than quantized EAGLE on an A10G GPU, with reduced VRAM usage and better effectiveness.

Limitations

Our focus is primarily on memory-efficient and training-efficient accelerations, so the speedups of our model may not be optimal when compared to other models based on the same target model. Also, we adopt the HuggingFace official quantization for its data-free calibration implementation (Dettmers et al., 2022), which may be subject to future performance improvements, potentially reducing the impact of quantization penalties. Our scope of application may also be limited to edge devices or in budget-sensitive environments where GPU memory is a major concern. Lastly, due to resource constraints, we are unable to extensively explore all popular LLMs of different sizes, so our proposed formula for predicting acceptance rates in Eq. 10 may need further adaptation for other models.

References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Caio César Teodoro Mendes, Weizhu Chen, Vishrav Chaudhary, Parul Chopra, Allie Del Giorno, Gustavo de Rosa, Matthew Dixon, Ronen Eldan, Dan Iter, Amit Garg, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Jamie Huynh, Mojan Javaheripi, Xin Jin, Piero Kauffmann, Nikos Karampatziakis, Dongwoo Kim, Mahoud Khademi, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Chen Liang, Weishung Liu, Eric Lin, Zeqi Lin, Piyush Madan, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Xia Song, Masahiro Tanaka, Xin Wang, Rachel Ward, Guanhua Wang, Philipp Witte, Michael Wyatt, Can Xu, Jiahang Xu, Sonali Yadav, Fan Yang, Ziyi Yang, Donghan Yu, Chengruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *Preprint*, arXiv:2404.14219.

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2024. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. 2024. Hydra: Sequentially-dependent draft heads for medusa decoding. *Preprint*, arXiv:2402.05109.

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *JETC*.

Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. 2023. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. In *EMNLP*.

Nikhil Bhendawade, Irina Belousova, Qichen Fu, Henry Mason, Mohammad Rastegari, and Mahyar Najibi. 2024. Speculative streaming: Fast LLM inference without auxiliary models. *Preprint*, arXiv:2402.11131.

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. *Preprint*, arXiv:2401.10774.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

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

Tri Dao. 2024. Flashattention-2: Faster attention with better parallelism and work partitioning. In *ICLR*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2019. Universal transformers. *Preprint*, arXiv:1807.03819.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *NeurIPS*.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A Aly, Beidi Chen, and Carole-Jean Wu. 2024. LayerSkip: Enabling early exit inference and self-speculative decoding. *Preprint*, arXiv:2404.16710.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *ICLR*.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of LLM inference using lookahead decoding. *Preprint*, arXiv:2402.02057.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. 2024. Better & faster large language models via multi-token prediction. *Preprint*, arXiv:2404.19737.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *NIPS*.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *ICLR*.

Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. 2024. CLLMs: Consistency large language models. In *ICML*.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander Rush. 2021. Block pruning for faster transformers. In *EMNLP*.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. Eagle: Speculative sampling requires rethinking feature uncertainty. In *ICML*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. ACL.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. 2024. QServe: W4A8KV4 quantization and system co-design for efficient LLM serving. *Preprint*, arXiv:2405.04532.

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

James Liu, Guangxuan Xiao, Kai Li, Jason D. Lee, Song Han, Tri Dao, and Tianle Cai. 2024b. Bitdelta: Your fine-tune may only be worth one bit. *Preprint*, arXiv:2402.10193.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *ICLR*.

Christos Louizos, Max Welling, and Diederik P. Kingma. 2018. Learning sparse neural networks through L-0 regularization. In *ICLR*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. LLM-pruner: On the structural pruning of large language models. In *NeurIPS*.

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 *ASPLOS*.

Tycho FA Ouderaa, Markus Nagel, Mart van Baalen, Yuki M Asano, and Tijmen Blankevoort. 2024. The LLM surgeon. In *ICLR*.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: memory optimizations toward training trillion parameter models. In *ACM Supercomputing Conference*.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. 2024. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *Preprint*, arXiv:2404.02258.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2023. On the effect of dropping layers of pre-trained transformer models. *Comput. Speech Lang.*

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 *ACL*.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. 2023. Consistency models. In *ICML*.

Yang Song, Chenlin Meng, Renjie Liao, and Stefano Ermon. 2020. Accelerating feedforward computation via parallel nonlinear equation solving. In *ICML*.

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

Mitchell Stern, Noam M. Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *NeurIPS*.

Xin Sun, Tao Ge, Furu Wei, and Houfeng Wang. 2021. Instantaneous grammatical error correction with shallow aggressive decoding. In *ACL*.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. *ICPR*.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing. *Preprint*, arXiv:1910.03771.

Haoyi Wu and Kewei Tu. 2024. Layer-Condensed KV Cache for efficient inference of large language models. *Preprint*, arXiv:2405.10637.

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 *EMNLP*.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *Preprint*, arXiv:2401.07851.

Mengzhou Xia, Zexuan Zhong, and Danqi Chen. 2022. Structured pruning learns compact and accurate models. In *ACL*.

Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. 2024. Multi-candidate speculative decoding. *Preprint*, arXiv:2401.06706.

Chen Zhang, Zhuorui Liu, and Dawei Song. 2024. Beyond the speculative game: A survey of speculative execution in large language models. *Preprint*, arXiv:2404.14897.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2023a. Draft & Verify: Lossless large language model acceleration via self-speculative decoding. *Preprint*, arXiv:2309.08168.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023b. Adaptive budget allocation for parameter-efficient fine-tuning. In *ICLR*.

Weilin Zhao, Yuxiang Huang, Xu Han, Chaojun Xiao, Zhiyuan Liu, and Maosong Sun. 2024. Ouroboros: Speculative decoding with large model enhanced drafting. *Preprint*, arXiv:2402.13720.

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 *NeurIPS*.

Appendix

A Training Configurations

We train 7B models with a per-device batch size of 2, a max sequence length of 2048, and 8 gradient accumulation steps with 4 GPUs. For 13B models, we use a per-device batch size of 1, a max sequence length of 1024, and the same 8 gradient accumulation steps but with 8 GPUs parallelized in Zero-3 using DeepSpeed with parameter offloading (Rajbhandari et al., 2020).

We uniformly apply a learning rate of $3 \cdot 10^{-5}$ with 50 linear warm-up steps and a fused AdamW (Loshchilov and Hutter, 2019) kernel from PyTorch for optimization. During our multi-token prediction training, we sample $20,000$ data rows and mask out 15% tokens randomly.

B S3D using LoRA

Using an S3D model fine-tuned with adapters can easily guarantee the same output as the original target model, thus maintaining the original model quality. With this intention, we have trained S3D models using LoRA (Hu et al., 2022) with the same amount of data as full fine-tuning.

We apply LoRA to the optimal layer skipping schemes ($5 \rightarrow 26$ and $7 \rightarrow 24$) and vary matrix ranks r from 12 to 128, and LoRA parameter α from 32 to 128. However, as shown by Figure 8, the speed of a LoRA-based S3D model is notably penalized by the overheads from the attached linear adapters, and the acceptance rate underperforms compared to that of a fully fine-tuned model. As a result, we observe inferior speed when using S3D with LoRA adapters. We have also experimented AdaLoRA (Zhang et al., 2023b) but have observed similar negative results for end-to-end speeds.

Nevertheless, we have seen that the BitDelta method (Liu et al., 2024b) successfully compressed fine-tuned adapter weights by more than 10x. However, BitDelta requires large training resources and needs to keep three model copies simultaneously. Despite this, we still see great potential to achieve similar efficiency to a fully fine-tuned S3D model with guaranteed original model outputs using adapters, but we leave this to future work.

C Phi-3 S3D Ablation Study

To understand the high cost-effectiveness of our proposed S3D speculative decoding scheme when applied to the Phi-3 target model, we conducted an ablation study, analyzing the contributions of the target model, quantization, and the S3D method.

From Table 4, it is observed that avoiding quantization generally only enhances speed, while the significant improvements in cost-effectiveness stem from two main factors: 1) transitioning to Phi-3 Mini as the target model, and 2) implementing the S3D method. However, the latter is crucial for surpassing the state-of-the-art SD in both speedups and cost-effectiveness under limited memory.

D Issues of CLLMs

Similar to our method, CLLMs (Kou et al., 2024) incurs no additional memory costs to the original model. It iterates and verifies multiple tokens in parallel using Jacobi iterations with random initial guess tokens. Consequently, in Table 1 and 2, CLLM has achieved higher speed-memory ratios comparable to ours.

However, we have discovered that the CLLM model we have evaluated is prone to generating repetitive patterns, presumably due to its training objectives, which encourage the target model to shorten the Jacobi trajectories before reaching the fixed point. While this can accelerate inference convergence and mitigate the issue of relatively slow speedups from vanilla Jacobi decoding, we have observed relatively low effectiveness scores from the officially trained 7B model using ShareGPT data (shown in Table 1). In contrast, S3D preserves most of the effectiveness of the original model.

In Figure 7, we present a comparison between two example outputs of CLLM. The one with repetitive patterns can generate tokens 2x faster than its counterpart, raising questions about the speed optimization of CLLM in terms of preserving the effectiveness of the original model.

E Prediction Formula Details

Our prediction for both acceptance rates and speeds shown in Fig 5 can be captured by only a few hyperparameters. For Eq. 10, we use $U = 0.01$. For the discount function of the k-th token acceptance rate, we use a linear interpolation of real data from the MT-Bench experiments shown in Figure 10. Specifically,

$$\alpha_k(\beta) = (1.22 - 0.22k) \cdot \alpha(\beta) \qquad (14)$$

for $k \leq 5$. Similarly, we profile the iteration overheads and set $\delta = 0.04$ in Eq. 12.

Table 2: The cost-effectiveness comparisons on a low-end RTX 3060 GPU considering peak VRAM costs. All models are 8-bit quantized and are based on the 7B LLaMA-v2 target model except mentioned otherwise in parentheses. The largest three numbers in each column are highlighted in *italics* or **bold**. "Peak" denotes the peak VRAM usage in GiB. Effectiveness metrics are omitted here as they mirror those in Table 1.

Model \ Metric	**MT-Bench**			**Human-Eval**			**CNN-Daily**			**Overall**
	Peak $\downarrow$	**Tok / s**	M	**Peak** $\downarrow$	**Tok / s**	M	**Peak** $\downarrow$	**Tok / s**	M	M
Baseline	8.42	8.11	1.00	*8.35*	7.82	1.00	*8.90*	7.42	1.00	1.00
Self-Spec.	7.81	5.71	0.76	7.46	5.46	0.78	8.08	5.68	0.84	0.79
Medusa (Vicuna)	*9.09*	10.32	1.18	*8.94*	11.97	1.43	*9.36*	7.99	1.02	1.21
EAGLE	*9.59*	**15.18**	1.64	*9.45*	**17.39**	1.97	*9.86*	**13.82**	**1.68**	1.76
MCSD	7.78	8.51	1.14	7.42	8.46	1.22	8.07	7.01	1.04	1.13
Sequoia	*8.44*	10.29	1.27	*8.35*	10.48	1.34	8.57	9.01	1.26	1.29
Ouroboros	7.94	6.28	0.82	7.60	6.79	0.95	8.23	5.62	0.82	0.86
CLLM	7.47	**14.14**	**1.97**	7.39	**19.19**	**2.77**	7.53	9.28	**1.48**	**2.07**
Ours										
S3D	7.81	13.99	**1.86**	7.58	14.99	**2.11**	8.72	**10.41**	1.43	**1.80**
S3D (Phi-3, fp16)	8.15	**29.88**	**3.23**	7.92	**33.70**	**3.18**	8.81	**19.49**	**2.65**	**3.23**

Table 3: The cost-effectiveness comparison considering peak VRAM costs for selected models using 13B 8-bit quantized LLaMA as the target model (A10G GPU). The largest number in each column are highlighted in *italics* or **bold**. "Peak" denotes the peak VRAM usage in GiB. The overall (averaged) results count for both M speed and the relative effectiveness metrics compared to the baseline.

Model \ Metric	**MT-Bench**				**Human-Eval**				**CNN-Daily**					**Overall**	
	Peak $\downarrow$	**Tok / s**	M	**Score**	**Peak** $\downarrow$	**Tok / s**	M	**Acc. %**	**Peak** $\downarrow$	**Tok / s**	M	**R-1**	**R-L**	M	**Eff.**
Baseline (13B)	15.39	5.60	1.00	7.36	15.03	5.51	1.00	6.10	15.88	5.09	1.00	0.21	0.15	1.00	1.00
EAGLE (13B)	*16.96*	10.67	1.73	**7.40**	*16.82*	**12.58**	2.04	4.27	*17.29*	**9.82**	**1.77**	0.21	0.15	1.85	0.90
S3D (13B)	14.39	**10.70**	**2.04**	6.18	14.08	12.23	**2.37**	**6.71**	15.80	7.87	1.55	**0.25**	**0.17**	**1.99**	**1.04**

Table 4: Ablations on the cost-effectiveness of S3D models based on Phi-3 Mini (A10G GPU), considering peak VRAM costs. The largest two numbers in each column are highlighted in *italics* or **bold**. "Peak" denotes the peak VRAM usage in GiB. Although using non-quantized fp16 generally improves model speeds, it does not neccessarily add cost-effectiveness due to the additional memory overheads.

Models \ Metrics	**MT-Bench**			**Human-Eval**			**CNN-Daily**			**Overall**
	Peak $\downarrow$	**Tok / s**	M	**Peak** $\downarrow$	**Tok / s**	M	**Peak** $\downarrow$	**Tok / s**	M	M
LLaMA (8-bit)	*8.53*	7.02	1.00	*8.43*	7.02	1.00	*8.96*	6.53	1.00	1.00
Phi-3 (8-bit)	4.64	8.91	2.33	3.78	8.82	2.80	5.49	7.99	2.00	2.38
Phi-3 (fp16)	7.99	**16.43**	2.50	8.22	15.97	2.33	8.80	**13.59**	2.12	2.32
Phi-3 (S3D, 8-bit)	4.79	14.39	**3.65**	5.11	**26.71**	**6.28**	5.53	11.11	**2.76**	**4.23**
Phi-3 (S3D, fp16)	*8.19*	**26.00**	**3.86**	*8.56*	**48.23**	**6.77**	*8.88*	**16.95**	**2.62**	**4.42**

USER: tell me a few interesting facts about the sun and the moon.

ASSISTANT: 1. The sun is approximately 4.6 million miles away from Earth,while the moon is about 238,900 miles away. 2. The sun is about 864,000 miles in diameter, while the moon is only about 2,159 miles in diameter. 3. The sun is made up of hydrogen and helium, while the moon is primarily composed of silicon and iron. 4. The sun is a spherical object, while the moon has a slightly irregularshape due to its formation process. 5. The sun takes approximately 24 days to complete one rotation, while the moon takes approximately 27.3 days to complete one rotation around Earth. 6. The sun is approximately4.6 billion years old, while the moon is estimated to be around4.5 billion years old. 7. The sun's surface temperature is approximately 5,500°F (3,000°C), while the moon's surface temperature ranges from -273°F (-170°C) during the night to 260°F (127°C) during the day. 8. The sun is a star, while the moon is a natural satellite of Earth. 9. The sun is the source of all energy on Earth,while the moon has no intrinsic energy source. 10. The sun is the largest object in the solar system, while the moon is the largest natural satellite in the solar system.

USER: tell me a few interesting facts about the sun and the moon.

ASSISTANT: 1. The sun is a gigantic ball of plasma, while the moonis a rocky body. 2. The sun is approximately 4.6 billion years old, while the moon is approximately 4.5 billion yearsold. 3. The sun is approximately 109 times larger thanthe moon, and the moon is approximately 300 times smaller than thesun. 4. The sun is approximately 100 times more massivethan the moon. 5. The sun is approximately 93 million miles(150 million kilometers) away from Earth, while the moon isapproximately 238,900 miles (384,400 kilometers) away. 6. The sun is approximately 10,000 degrees Fahrenheit (5,538 degrees Celsius) at its surface, while the moon's surface temperature rangesfrom -273 to -233 degrees Fahrenheit (120 to -150 degrees Celsius). 7. Thesun is approximately 864,000 miles (1,392,000 kilometers) in diameter, while the moon isapproximately 2160 miles (3474 kilometers) indiameter. 8. The sun is approximately 109 times more luminous than the moon. 9. The sun is approximately 10,000 times more massive than the moon. 10. Thesun is approximately 10,000 times more luminous thanthe moon. 11. The sun is approximately 10,000 times more massive than the moon. 12. The sun isapproximately 10,000 times more luminous than the moon. 13. The sun is approximately 10,000times more massive than the moon. 14. The sun is approximately 10,000 times more luminous than the moon. 15. The sun is approximately 10,000 times moremassive than the moon. 16. The sun is approximately 10,000 times more luminous than the moon. 17. The sun is approximately 10,000 times more massive thanthe moon. 18. The sun is approximately 10,000 times more luminous than the moon. 19. Thesun is approximately 10,000 times more massive than the moon. 20. The sun is approximately 10,000times more luminous than the moon. **... (repetitive pattern continues)**

Figure 7: Examples of CLLM outputs (on a RTX 3060 GPU with 8-bit quantization). The implementation is non-deterministic due to potential numerical errors. As a result, we are able to show two different inputs from the same prompt. **Upper**: A good example which has a speed of 10.85 tokens per second. **Lower**: A bad example which shows repetitive patterns at the end, having a 2x higher speed of 21.98 tokens per second.

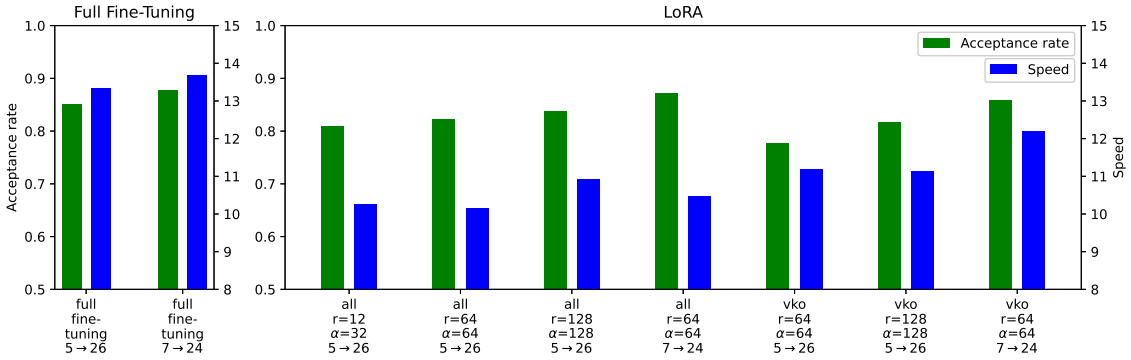

Figure 8: S3D full fine-tuning compared with using LoRA adapters (MT-Bench). LoRA settings include attaching adapters to all linear layers (all) or only attaching to value, key, and output projection layers (vko). Given the same layer skipping scheme, vko and low-rank LoRA have lower inference overheads but achieve lower acceptance rates (at depth-0). Overall, LoRA does not offer similar speeds compared to full fine-tuning, although it reliably maintains the original model output.

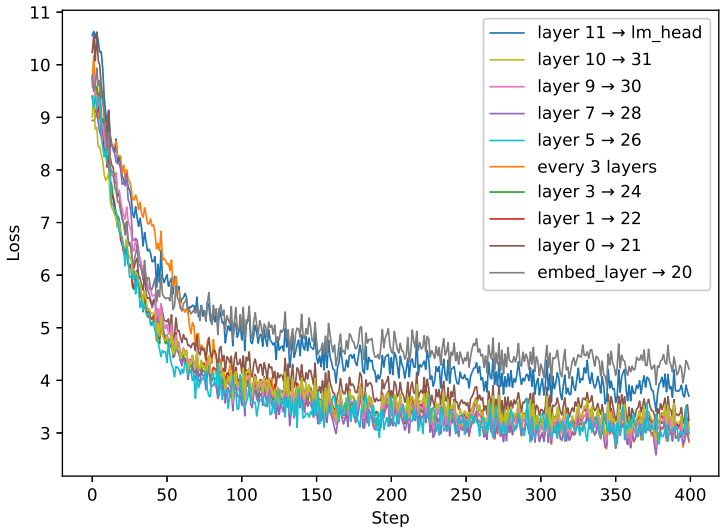

Figure 9: The training loss when fine-tuning different skipping schemes. We alter the skipped layers while keeping the total number of used layers (12 layers or $\beta = 0.375$) unchanged.

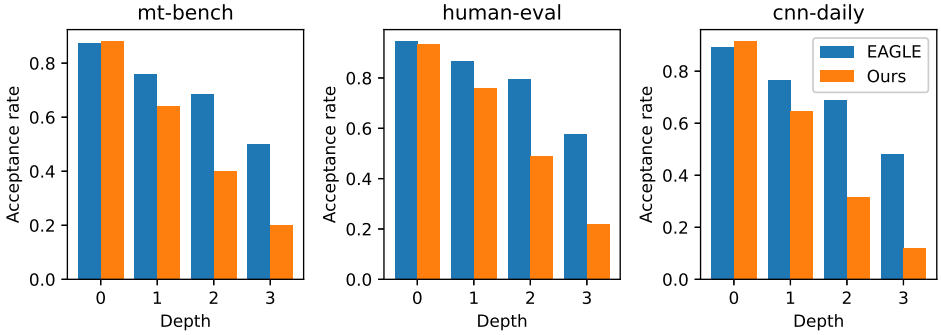

Figure 10: Acceptance rates comparison at different draft token tree depths, compared between ours (S3D) and the state-of-the-art open-source model EAGLE in different datasets. Since we predict tokens simultaneously, our acceptance rates drop more at future positions. However, we achieve similar acceptance rates at the draft root and are able to outperform EAGLE cost effectively while using much less training data.