Scaling LLM Speculative Decoding: Non-Autoregressive Forecasting in Large-Batch Scenarios

Luohe Shi[1], Zuchao Li[2*], Lefei Zhang[1], Baoyuan Qi[3], Guoming Liu[3], Hai Zhao[4]

[1]School of Computer Science, Wuhan University
[2]School of Artificial Intelligence, Wuhan University
[3]Xiaomi
[4]School of Computer Science, Shanghai Jiao Tong University
{shiuohe, zcli-charlie, zhanglefei}@whu.edu.cn
{qibaoyuan, liuguomin}@xiaomi.com
zhaohai@cs.sjtu.edu.cn

Abstract

Speculative decoding accelerates LLM inference by utilizing otherwise idle computational resources during memory-to-chip data transfer. Current speculative decoding methods typically assume a considerable amount of available computing power, then generate a complex and massive draft tree using a small autoregressive language model to improve overall prediction accuracy. However, methods like batching have been widely applied in mainstream model inference systems as a superior alternative to speculative decoding, as they compress the available idle computing power. Therefore, performing speculative decoding with low verification resources and low scheduling costs has become an important research problem. We believe that more capable models that allow for parallel generation on draft sequences are what we truly need. Recognizing the fundamental nature of draft models to only generate sequences of limited length, we propose SpecFormer, a novel architecture that integrates unidirectional and bidirectional attention mechanisms. SpecFormer combines the autoregressive model's ability to extract information from the entire input sequence with the parallel generation benefits of non-autoregressive models. This design eliminates the reliance on large prefix trees and achieves consistent acceleration, even in large-batch scenarios. Through lossless speculative decoding experiments across models of various scales, we demonstrate that SpecFormer sets a new standard for scaling LLM inference with lower training demands and reduced computational costs.

Code — hhttps://github.com/ShiLuohe/SpecFormer

Introduction

Large language models (LLMs) that utilizes Transformer Decoders have rapidly become the industry standard in recent years, owing to their favorable properties such as scalability in training and lossless handling of long-context dependencies (OpenAI 2023). Nevertheless, these models continue to follow the conventional sequence-to-sequence generation paradigm: autoregressive decoding. Autoregressive decoding refers to the process where tokens are generated one at a time; each newly generated token is fed back into the model as input for the next step, alongside the existing context, to perform another forward pass. This paradigm offers several notable advantages. With only a causal mask, the attention mechanism can be readily adapted for generation tasks, making training straightforward. It also allows for the generation of virtually unlimited-length outputs and enables acceleration through state caching for repeated input prefixes (Shi et al. 2024), and some corresponding acceleration opportunities (Guo et al. 2025; Yang et al. 2025; Zhao et al. 2025b,a; Tang et al. 2025).

However, during inference, generating one token at a time results in low arithmetic intensity (AI, Williams, Waterman, and Patterson 2009). Once a datum fetched from memory into the chip, it contributes to only a few operations. In contrast, chips can perform hundreds of operations in the time, leading to substantial underutilization of compute resources. On the infrastructure side, techniques such as prefill-decoding (PD) separation (Zhong et al. 2024) and continuous batching (Yu et al. 2022; Kwon et al. 2023) have been introduced to improve overall compute utilization and user experience. Nonetheless, under constraints imposed by service-level objects (SLOs, Wang et al. 2024), these techniques often fall short of leveraging the full computational potential. Increasing AI, the ratio of computation to data transfer, is the fundamental approach to enhancing hardware efficiency during generation.

Speculative decoding (SD, Xia et al. 2024) is one of the most effective approaches for improving AI. Its core idea is to generate multiple tokens per pass of the large model. The process consists of three main steps (Stern, Shazeer, and Uszkoreit 2018): 1. **Multi-token generation**: Based on information from the previous forward pass, the model samples multiple draft tokens. 2. **Multi-token verification**: The model evaluates all draft tokens simultaneously to determine whether each one aligns with its own top prediction, while also extracting and storing information for the next round of multi-token generation. 3. **Multi-token acceptance**: The model decides whether to accept the draft tokens based on the verification results and accordingly updates the contex-

*Corresponding Author

tual information.

Multi-token generation is the most critical component of SD, as the acceptance rate of the sampled drafts directly determines how effectively computational resources are utilized. **In this work, we focus specifically on lossless SD**, which adheres to two strict conditions: 1. Only draft tokens that exactly match the outputs of the large model are accepted. 2. The LLM itself must remain unmodified. These constraints make a purely acceleration-oriented SD, ensuring strict mathematical equivalence with the original model outputs.

A key observation about SD is that it does not reduce (usually increases significantly) the total amount of computation. The acceleration arises from repurposing compute capacity that would otherwise be idle while waiting for data transfer. In other words, every SD-based method has a theoretical upper bound on speedup, corresponding to the full utilization of previously wasted compute. It is important to note that continuous batching is also a method for reducing idle compute. Consequently, in batched settings where unused compute capacity is already diminished, speculative decoding methods face a stricter efficiency requirement.

Current SD methods can be broadly categorized into autoregressive and non-autoregressive approaches (Hu et al. 2025). They all generate draft tokens in time that scales linearly with the number of draft tokens, whether through autoregression or by accessing different parameters.

However, we've noticed a significant issue with current methods: their incompatibility with large batch sizes. A larger batch size means each parameter experiences higher computational intensity, as it's read and reused multiple times. This, in turn, reduces the computational cycles available for draft tokens. In other words, under batch processing conditions, there are twice as many draft tokens to process in less available time. This point is illustrated in Figure 1, where we show that when processing the same number of tokens while varying the batch size, the speed at which peak computational power is reached accelerates with increasing batch size.

The number of draft tokens we can use is less than what's needed to reach peak performance. This means that in a batched environment, the usable draft tree size is rapidly compressed. Given that online batching has been widely adopted by mainstream inference frameworks, current speculative decoding methods must adapt to scarcer resources. This implies they can no longer rely on traditional, massive draft trees, but instead need to focus on higher accuracy drafts themselves. Mainstream models face a significant challenge in this regard: they struggle to efficiently scale to larger sizes to displace capabilities. This is due to their excessive position-dependent parameters—whether these are parameters that need to be repeatedly accessed for each position in AR methods, or parameters separately allocated for each position in non-AR methods. The problem with position-dependent parameters is that scaling up exponentially increases additional costs because multiple tokens are required in a sequence. This means these methods not only inherently use more computational power but are also highly sensitive to changes in available compute.

Therefore, we aim to improve the performance of SD under low draft token budgets, by directly enhancing the capability of the draft generation model. This enables SD to be effectively applied in batched inference settings. To avoid fine-tuning the original LLM, the draft model must receive sufficiently rich input information. To this end, we employ a context causal attention to extract contextual information from the hidden states of the input sequence. We observe that in traditional approaches, the parameters used for draft generation are position-dependent, i.e., generating each position in the draft sequence typically requires accessing a large number of parameters tied to that specific position. Instead, we seek a prediction mechanism in which the majority of parameters are position-independent, while retaining only a limited amount of positional information. Furthermore, we identify a key distinction between draft generation in SD and open-ended generation in LLMs: SD only requires a small number of future tokens, rather than unbounded generation. Motivated by this, we adopt a Draft Bi-directional Attention architecture for draft token generation. This forms the basis of our proposed SpecFormer architecture.

More specifically, we enable the model to efficiently extract information from both greater depth and breadth simultaneously through multi-level hidden state feature fusion combined with a causal masked attention mechanism. Subsequently, by assigning a specific set of matrix multiplication weights to each position, we can effectively inject positional information to obtain the initial state for each subsequent token. Finally, a standard Encoder layer is used to parallelize the fine-grained generation of tokens at each subsequent position.

We evaluate our proposed method on models of approximately 4B, 7B, and 14B parameters (Yang et al. 2024), conducting both theoretical and real-world experiments. In the theoretical experiments, we constrain the number of draft tokens to simulate varying levels of redundant computational capacity and draft model cost. Under these conditions, we measure the average accepted token length across different methods to assess their efficiency. In the real-world experiments, we evaluate the acceleration ratio of our method under different batch size settings using dialogue datasets and standard benchmarks, demonstrating its effectiveness in practical deployment scenarios.

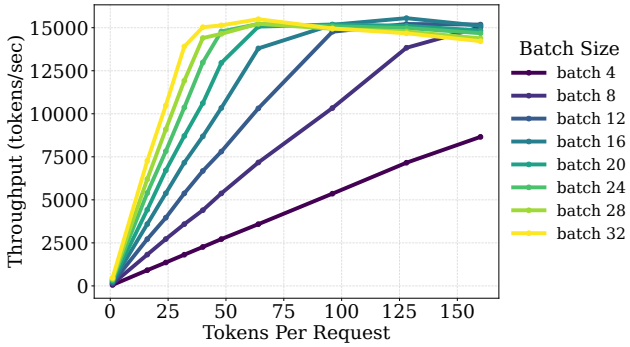

Figure 1: How batch size affects the max draft size.

Background and Related Works

Non-autoregressive SD Approaches

Non-autoregressive methods refer to SD algorithms in which the draft tokens are generated without causal dependencies among them. The most common examples include Multi-Token Prediction (MTP, Gloeckle et al. 2024) and Medusa (Cai et al. 2024). These approaches share a common principle: leveraging the last hidden state (LHS) of the LLM, originally used for predicting the next token, to predict multiple future tokens simultaneously. Medusa trains a separate MLP layer for each target position, projecting the LHS into a new token space, which is then fed into the LM_Head to generate the corresponding draft token. In contrast, MTP designs multiple LM_Heads, each dedicated to generating the draft token at a specific future position. Positional-sharing parameters are typically less while Positional-specific ones remains fairly many for these methods. These methods often suffer from limited predictive capacity due to their inability to access information from the entire sequence, and they typically require fine-tuning the entire model.

Autoregressive SD Approaches

Autoregressive methods employ a smaller sequence model to generate future tokens autoregressively based on the input sequence. Autoregressive decoding can operate at three levels:

1. **Token level**: These methods use a standalone small language model (SLM) to generate future tokens autoregressively. The SLM typically shares the same vocabulary as the LLM. It receives the input tokens from the LLM, samples several future tokens autoregressively, and then passes them to the LLM for validation. A key advantage is that, if a suitable SLM exists, no additional training is required. However, such models are difficult to obtain, and the approach introduces significant KV cache overhead. A representative method is BiLD (Kim et al. 2023) decoding (Xia et al. 2023; Huang, Guo, and Wang 2024; Zhou et al. 2024; Bachmann et al. 2025).
2. **LHS level**: These methods perform autoregressive decoding over LHS representations. A small model consumes the LHS output from the LLM, predicts the next LHS, and recursively feeds it into itself. The resulting LHSs are then converted to token predictions and validated by the LLM. The small model is typically a decoder layer and requires additional training, but since the LLM itself is not modified, the training cost in both time and memory is significantly lower than fine-tuning. The primary limitation lies in the difficulty of aligning the small model to the LHS space, which can impair its performance. Representative methods include EAGLE (Li et al. 2025b), HASS (Zhang et al. 2025), Deepseek-V3 MTP (DeepSeek-AI 2024), etc.(Gao et al. 2025; Chen et al. 2025)
3. **Independent representation** : These methods construct a separate latent space by combining the LLM's LHS with auxiliary information, and perform autoregressive decoding in this space. A notable example is EAGLE-3 (Li et al. 2025a).

A common challenge across autoregressive decoding models is that the repeated invocation of the small model means that, even with identical content, its parameters remain position-dependent, leading to higher computational costs. Furthermore, due to the limited capacity of the small model, these methods often require a very wide prefix tree to explore multiple hypotheses in parallel, in order to achieve acceptable prediction accuracy.

Methods

From Arithmetic Intensity to SD Evaluation

Arithmetic intensity (AI) is defined as the ratio between the number of required floating-point operations and the number of bytes of data that must be read. For a model with M parameters operating in half-precision, the arithmetic intensity AI_m is given in Equation 1.

$$\begin{aligned} AI_m &= \frac{\text{Model FLOPS}}{\text{Memory I/O}} \\ &= \frac{2 \cdot M}{\text{bytes}(\mathbf{bf16}) \cdot M} = 1 \end{aligned} \tag{1}$$

For an acceleration chip, we can estimate the ideal arithmetic intensity AI_c required to fully utilize its compute capacity by examining the ratio of its peak FLOPs to memory bandwidth (typically DDR, GDDR, or HBM). For Tesla A100-80G, the AI_c as shown in Equation 2.

$$\begin{aligned} AI_c(\text{A100}) &= \frac{\text{Peak FLOPS } \mathbf{bf16}}{\text{Memory Bandwidth}} \\ &= \frac{311.84 \text{ TFLOPS/s}}{2.04 \text{ TB/s}} = 152.86 \end{aligned} \tag{2}$$

We define the redundancy ratio ρ as the ratio AI_c/AI_m, which represents both the ideal batch size and the theoretical upper bound of speedup achievable through batching effects. It should be noted that due to practical factors such as scheduling overhead, ρ does not reflect actual performance precisely, but it provides a useful baseline for system-level analysis. We prefer smaller values of ρ, as a lower ρ indicates less wasted compute, with $\rho = 1$ representing the ideal case where no redundancy remains.

Previous work on SD has typically focused on average accepted token length, i.e., the average number of tokens accepted per invocation of the LLM. However, we argue that this metric is overly coarse-grained: it obscures the underlying total computational cost, making it difficult to adapt methods to new deployment scenarios. We contend that a better criterion for evaluating an SD method is to examine its performance under a fixed draft token budget.

For a given SD algorithm, suppose it increases the total computation by a factor of p, generates k draft tokens per step, and among them, an average of a tokens are accepted. Then, the effective AI gain relative to standard LLM decoding which we want to maximize, denoted as r_1, and the on-chip AI gain, denoted as r_2, can be derived as a function of ρ in Equation 3, with bs representing the batch size.

$$\max r_1 = \frac{a}{p} AI_m, \quad \mathbf{s.t.}\ r_2 = k \leq \frac{\rho}{bs} \tag{3}$$

Moreover, if the SD model requires m_p parameters to generate draft token of each position, and m_s parameters that shares within all positions, with the draft sequence length l_d, the p is given in Equation 4.

$$p = 1 + \frac{m_s + l_d \cdot m_p}{M} \tag{4}$$

Finally, we define an optimization coefficient κ in Equation 5, which captures the model's ability to accelerate under constrained resources. We aim to maximize κ, or increase the draft token acceptance rate while minimizing the computational overhead of the draft model. In prior work, k was often either ignored or fixed to a relatively large constant, owing to the availability of abundant redundant compute. However, as the batch size increases, the available redundant compute rapidly diminishes, making k a critical factor that significantly impacts performance.

$$\kappa = \frac{a \cdot l_d}{k} \tag{5}$$

General Notations

We define $\mathcal{C}$ as our training corpus with $|\mathcal{C}|$ entries. An entry $c \in \mathcal{C}$ is a list a tokens $x_1 x_2 \dots x_{|c|}$. The training goal of next-token prediction for LLM pretraining is to find the $\theta_{\mathcal{LM}}$ that minimize the cross entropy loss, given in Equation 6a. For an SD module with parameters $\theta_{\mathcal{SD}}$ and maximum drafting length l_d, the optimizing goal is given in Equation 6b.

$$\arg\min_{\theta_{\mathcal{LM}}} \sum_{c \in \mathcal{C}} \sum_{i=2}^{|c|} \frac{-\log P_{\theta_{\mathcal{LM}}}\left(x_i \mid x_1 \dots x_{i-1}\right)}{|\mathcal{C}| \cdot |c|} \tag{6a}$$

$$\arg\min_{\theta_{\mathcal{SD}}} \sum_{c \in \mathcal{C}} \sum_{j=2}^{l_d+1} \sum_{i=1+j}^{|c|} \frac{-\log P_{(\theta_{\mathcal{LM}}, \theta_{\mathcal{SD}})}\left(x_i \mid x_1 \dots x_{i-j}\right)}{|\mathcal{C}| \cdot |c| \cdot l_d} \tag{6b}$$

We further denote L as the layer count of the base LLM and d_h as the hidden size. Hidden states $\mathrm{HS} \in \mathbb{R}^{(L+1)\times|c|\times d_h}$ are the states that traversing between layers, where $\mathrm{HS}[i]$ represents the i-th layer of HS. Specifically, $\mathrm{HS}[0] = \mathrm{Embedding}(c)$ and $\mathrm{LHS} = \mathrm{HS}[L]$.

Finally, we define Equation 7 to simplify the description of pre-norm residual connected units.

$$(\mathrm{Ops} \cdot \mathrm{Norm} + \mathbb{I})(X) \iff \mathrm{Ops}(\mathrm{Norm}(X)) + X \tag{7}$$

SpecFormer

Our proposed SpecFormer comprises a Context Causal Attention and a Draft Bi-directional Attention, depicted in Figure 2a, incorporating both unidirectional and bidirectional attention along two dimensions, as shown in Figure 2b.

Context Causal Attention The Context Causal Attention module consists of three components: Hook and Downsampler, Causal Attention, and a Positional Feedforward Network (Positional FFN). Each component takes inputs passed through root-mean-square normalization (RMS Norm), with the Downsampler employing Grouped RMS Norm and the Causal Attention module utilizing a residual connection.

The hook module extracts information from the HS, following the approach introduced by Li et al.. Specifically, we select four layers: $\mathrm{HS}[0]$, $\mathrm{HS}[L/2]$, $\mathrm{HS}[L-1]$, and $\mathrm{HS}[L]$, and concatenate them to form a tensor $I \in \mathbb{R}^{bs\times|c|\times 4\times dh}$. We chose this more complex distribution because we noticed that hidden state representations at different layers often contain distinct information. Specifically, the last layer is directly used for predicting the next token, while the second-to-last layer typically encodes the most abstract information about the current token. The zeroth layer represents the embedding layer, which contains context-unprocessed token information. Finally, we included half the depth of layers as a supplement. We apply Grouped RMS Norm over the last dimension, assigning a group of scale parameters (initialized to 1) to each slice along the second-to-last dimension. The normalized tensor is then reshaped into $I_{\mathrm{Cat}} \in \mathbb{R}^{bs\times|c|\times 4d_h}$, which serves as the input to the Downsampler, a linear module with weights of W_{D} and no bias. The output $I_{\mathrm{D}} \in \mathbb{R}^{bs\times|c|\times d_h}$ is RMS-normalized again before being passed into the masked self-attention (MSA), which can be viewed as an additional $(L+1)$-th layer of the LLM. This design allows for easy integration with existing KV cache management frameworks. Formalized in Equation 8.

$$\begin{aligned} I_{\mathrm{D}} =& (\mathrm{MSA} \cdot \mathrm{RMS} + \mathbb{I})\left(W_{\mathrm{D}} \cdot I_{\mathrm{Cat}}\right) \\ I_{\mathrm{Cat}} =& \mathrm{GroupRMS}\left(\mathrm{HS}[0, L/2, L-1, L]\right) \end{aligned} \tag{8}$$

The Positional FFN is a linear projection from dimension d_h to $l_d \cdot d_h$ with weights W_{P}, effectively decomposing a representation into l_d position-specific components with added biases b_{P}. We argue that position-specific information in draft tokens should not be too simplistic, such as assigning a basic mask per position, yet using a full MLP for each position would be overly redundant. Therefore, we adopt this middle-ground approach. The number of position-related parameters is $l_d \cdot d_h^2$, which, while still quadratic in d_h, is smaller than methods like Medusa, which require at least $8 \cdot l_d \cdot d_h^2$, placing our method on the more efficient end. The output of the Context Causal Attention stage is a tensor $D \in \mathbb{R}^{bs\times|c|\times l_d\times d_h}$. Formalized in Equation 9.

$$D = W_{\mathrm{P}} \cdot \mathrm{RMS}(I_{\mathrm{D}}) + b_{\mathrm{P}} \tag{9}$$

Draft Bi-directional Attention The Draft Bi-directional Attention layer applies self-attention mechanisms within the draft token sequence, utilizing a standard self-attention (SA) module with residual connections and a Swish Gated Linear Unit (SwiGLU) feedforward network. All components are normalized using RMS Normalization. The output $E \in \mathbb{R}^{bs\times|c|\times l_d\times d_h}$. Formalized in Equation 10.

$$E = (\mathrm{SwiGLU} \cdot \mathrm{RMS} + \mathbb{I})((\mathrm{SA} \cdot \mathrm{RMS} + \mathbb{I})(D)) \tag{10}$$

It is important to emphasize that the attention mechanism operates along the draft token dimension; that is, for a sequence of length l_d, the effective batch size becomes $bs \cdot |c|$. In our implementation, we observed that FlashAttention 2 (Dao 2024; Dao et al. 2022) cannot handle batch sizes larger than 4095. To address this limitation, we partition the computation along the batch dimension, processing the attention in groups of 3072 samples per batch segment.

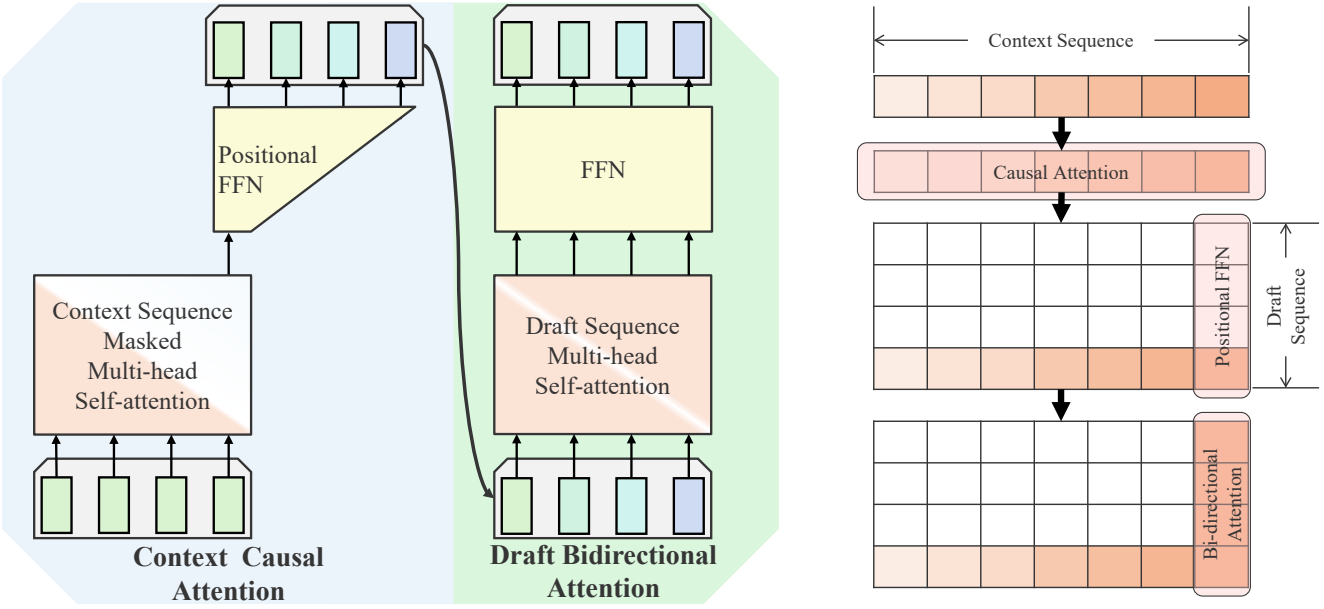

Figure 2: An overview of the proposed SpecFormer speculative decoding method.

Implementation Improvements

Efficient Grouped RMS Norm Through profiling, we found that RMS Normalization often becomes a performance bottleneck, primarily due to its significant consumption of CPU time slices. As a result, implementing Grouped RMS Norm with a loop-based approach tends to be inefficient. To address this, we customized a GPU kernel using Triton (Tillet, Kung, and Cox 2019) to implement the Grouped RMS Norm operation more efficiently.

Intra-batch Gradient Accumulation We adopted the gradient accumulation strategy around the LM Head as proposed by Gloeckle et al.. Specifically, for each position $j \in \{1, 2, \ldots, l_d\}$, we compute the loss sequentially, rather than simultaneously. This is because the vocabulary size in modern language models often exceeds 128K, making the softmax very expensive in storage. Instead, we sequentially map each position's hidden state to the vocabulary, compute gradients, and store them within the hidden states via backpropagation. Once gradients for all positions are computed, we continue the remaining backward pass together.

Experiment

Priliminary Experiment

We tested the GPU throughput achieved under different batch sizes and token counts to corroborate that the number of available tokens for speculative decoding is limited with large batch sizes. Our results are shown in Figure 1.

Setups

Training Corpus We trained our model on the UltraChat-200K (UC, Ding et al. 2023) dataset, which contains approximately 460K dialogue samples. Although the dataset itself is distilled from ChatGPT outputs, in our implementation, we opted to perform self-distillation (Zhang, Bao, and Ma 2022; Lasby et al. 2025) first. Specifically, we retained only the question parts from the original samples and regenerated the completions using the base LLM. This ensures that the distribution learned by the draft model strictly aligns with that of the base model. Our experiments demonstrate that this adjustment leads to performance improvements.

Base LLM We selected foundation models from the Qwen and LLaMA families, including Qwen2.5-3B, Qwen3-8B, Qwen3-14B, and LLaMA-3.1-8B. Unlike many previous works, we did not adopt the Vicuna (Zheng et al. 2023) series. This decision is based on two considerations: First, both the Vicuna model and its training dataset (ShareGPT) are relatively outdated. Second, as a chat model built on early versions of LLaMA (Touvron et al. 2023), Vicuna uses a small vocabulary (about 32K). Vocabulary size is closely correlated with the difficulty of token prediction in draft generation—larger vocabularies increase prediction difficulty. Modern models typically use vocabularies exceeding 128K, with some, such as Gemma (Kamath et al. 2025), reaching 256K, making Vicuna unrepresentative of current LLMs.

Evaluation Our evaluation set includes the test split of the UC dataset along with several popular benchmarks: MT-Bench (Zheng et al. 2023), HumanEval (Chen et al. 2021), GSM8K (Cobbe et al. 2021), Alpaca (Taori et al. 2023), and CNN/DM (See, Liu, and Manning 2017). For reporting purposes, we present averaged results across this combined set, as there is no strong evidence suggesting performance varies significantly across these datasets in our no-regression setting. Since we focus on lossless LLM acceleration, correctness is not a concern—the model's outputs remain identical

bs	k	W/o SD		HASS		EAGLE-3		Ours	
		κ	TPS	κ	TPS	κ	TPS	κ	TPS
1	4	1	41 (1×)	2.14	69 (1.70×)	2.16	70 (1.73×)	2.20	73 (1.78×)
	6	1	41 (1×)	2.17	71 (1.74×)	2.18	72 (1.75×)	2.22	74 (**1.81**×)
	8	1	41 (1×)	2.17	72 (1.75×)	2.19	72 (1.76×)	2.23	73 (1.80×)
4	4	1	162 (1×)	2.14	275 (1.70×)	2.16	277 (1.71×)	2.18	289 (1.78×)
	6	1	162 (1×)	2.17	282 (1.74×)	2.17	282 (1.73×)	2.22	293 (**1.81**×)
	8	1	162 (1×)	2.18	284 (1.75×)	2.18	279 (1.72×)	2.23	291 (1.80×)
16	4	1	681 (1×)	2.14	1164 (1.71×)	2.16	1175 (1.72×)	2.19	1212 (1.78×)
	6	1	681 (1×)	2.16	1190 (1.74×)	2.17	1185 (1.74×)	2.22	1233 (**1.81**×)
	8	1	681 (1×)	2.17	1189 (1.75×)	2.17	1192 (1.75×)	2.24	1220 (1.79×)
64	4	1	2590 (1×)	2.13	4454 (1.72×)	2.15	4429 (1.71×)	2.19	4610 (1.78×)
	6	1	2590 (1×)	2.17	4530 (1.75×)	2.17	4515 (1.74×)	2.22	4688 (**1.81**×)
	8	1	2590 (1×)	2.17	4541 (1.75×)	2.18	4507 (1.74×)	2.24	4610 (1.78×)
128	4	1	5143 (1×)	2.14	8800 (1.71×)	2.16	8846 (1.72×)	2.18	9154 (1.78×)
	6	1	5143 (1×)	2.16	8956 (1.74×)	2.17	8901 (1.73×)	2.22	9308 (**1.81**×)
	8	1	5143 (1×)	2.17	8945 (1.74×)	2.16	8845 (1.72×)	2.24	9206 (1.79×)

Table 1: The comparison between SpecFormer and baselines under different batch size and settings. The baseline methods may underperform compared to their reported values, as we impose a constraint on the draft token budget.

before and after acceleration.

Implementation Our method is implemented and trained using the *PyTorch* framework with few *Triton* and *FlashAttention* components. For inference, we leverage the *Medusa* decoding framework, as well as custom SD-compatible code based on the *HuggingFace Transformers* (Wolf et al. 2019). We conducted tests under various batch sizes, and report the theoretical speedup, efficiency factor κ, and actual speed gains. Detailed hyperparameters is given in Appendix 4.

Throughput Comparison

We constrain the available draft token budget to a relatively small value and then evaluate the system's throughput under varying batch sizes. We measure the throughput of our method using tokens per second (TPS), as shown in Table 1. We observe that our approach consistently outperforms the baseline methods. Notably, the baselines do not reach their reported performance levels in our setting because we constrain the available token budget to simulate scenarios with limited computational redundancy, such as those arising in large-batch inference. In contrast, our method achieves high throughput without relying on a large number of draft tokens, owing to its superior predictive capability. Furthermore, we evaluate the conversion rate from κ-to-TPS, and find that our method exhibits a higher conversion efficiency. This is primarily because our design adopts a non-autoregressive formulation, which results in higher arithmetic intensity and lower average per-token overhead, thereby improving overall efficiency.

Special Case Study

Self Distillation We evaluate the impact of self-distillation by comparing models trained with and without it on Qwen2.5-3B. Specifically, we first train an *No-Self-Distill* model using the original UC-200K dialogue dataset. Then, we apply self-distillation by retaining only the prompt side of each dialogue and generating completions using the base LLM, which are subsequently used to train the *Self-Distill* model. Notably, the self-distilled dataset is smaller in size, as it contains fewer dialogue turns.

The κ value and acceleration performance are reported in Table 2. We observe that without self-distillation, the model demonstrates negligible acceleration, as the learned token distribution does not originate from the base model, but rather from a different teacher model. While traditional distillation may partially mitigate this issue, we argue that self-distillation remains a necessary step, particularly in light of modern deployment frameworks like *vLLM*, which offer highly efficient offline inference and make strict alignment with the base model's output even more critical.

Base LLM Size To investigate the performance gains of our architecture under speculative decoding across different model sizes, we conducted experiments on the Qwen-3 series, including 4B, 8B, and 14B variants—covering a representative range of commonly used model scales. The acceleration results across these models are presented in Table 3. We also calculate the κ-to-TPS conversion ratio θ to measure how the draft module itself impact the efficiency.

We observe that as the model size increases, the predictor's ability to accurately guess future tokens are weakened, resulting in less acceleration gains. For instance, the 4B model achieves a speedup of 1.56×, whereas the 14B model sees a reduced speedup of 1.47×. However, we also find that larger models exhibit a more favorable θ, meaning that the relative overhead introduced by the predictor is smaller. This can be attributed to two main reasons: The increased number of layers in larger models leads to a smaller parameter

bs	k	W/o SD			No-Self-Distill			Self-Distill		
		l_d	κ	TPS	l_d	κ	TPS	l_d	κ	TPS
1	8	1	1	32 (1.00×)	8	1.19	30 (0.94×)	8	1.90	56 (1.76×)

Table 2: The comparison between to use or not to use self-distillation.

bs	k	Qwen3-4B			Qwen3-8B			Qwen3-14B		
		κ	TPS	θ	κ	TPS	θ	κ	TPS	θ
1	0	1	30 (1.00×)	1	1	31 (1.00×)	1	1	26 (1.00×)	1
	4	1.81	45 (1.50×)	1.21	1.74	45 (1.45×)	1.20	1.71	38 (1.46×)	1.17
	8	1.81	46 (1.54×)	1.18	1.76	46 (1.49×)	1.18	1.72	39 (1.46×)	1.18
4	0	1	147 (1.00×)	1	1	120 (1.00×)	1	1	105 (1.00×)	1
	4	1.84	224 (1.53×)	1.20	1.76	178 (1.48×)	1.19	1.71	157 (1.49×)	1.14
	8	1.86	227 (1.56×)	1.19	1.76	182 (1.49×)	1.18	1.72	154 (1.47×)	1.17
16	0	1	588 (1.00×)	1	1	488 (1.00×)	1	1	436 (1.00×)	1
	4	1.84	899 (1.53×)	1.20	1.76	726 (1.49×)	1.18	1.71	636 (1.47×)	1.16
	8	1.86	917 (1.56×)	1.19	1.77	726 (1.49×)	1.19	1.72	639 (1.46×)	1.18
64	0	1	2346 (1.00×)	1	1	1904 (1.00×)	1	1	1713 (1.00×)	1
	2	1.72	3435 (1.46×)	1.18	1.68	2734 (1.44×)	1.17	1.64	2454 (1.41×)	1.16
	4	1.84	3621 (1.53×)	1.20	1.75	2834 (1.48×)	1.18	1.71	2524 (1.47×)	1.16
128	0	1	4582 (1.00×)	1	1	3882 (1.00×)	1	1	3458 (1.00×)	1
	2	1.73	6725 (1.47×)	1.18	1.68	5586 (1.43×)	1.17	1.64	4834 (1.41×)	1.16
	4	1.84	7263 (1.53×)	1.20	1.75	5761 (1.48×)	1.18	1.71	5090 (1.47×)	1.16

Table 3: The comparison between our proposed method SpecFormer and baselines under size of base LLMs.

Method	SpecFormer	-Pos	+Att Mask	+Larger
κ	1.81	1.77	1.80	1.91

Table 4: Ablation Study

percentage for the predictor, and the larger weight matrices in big models dilute the overhead from scheduling. Overall, these results demonstrate that our method remains applicable across various model sizes, although it shows particularly strong benefits on smaller models.

Module Ablation Study

We conducted ablation studies on the Qwen3-4B model with $k = 8$. Our analysis focused on whether bidirectional attention improves the model's capability, whether applying a naïve linear transformation for positional encoding enhances performance, and the model's capability under a wider architecture. We modify each module individually and ran experiments accordingly, and the results are shown in Table 4.

We note the following: 1. Bidirectional attention improves the model but not substantial. However, considering that its impact on inference time is negligible, we chose to retain this structure. 2. The positional FFN contributes a large portion of improvement, which is expected as it accounts for a considerable amount of parameters. 3. Larger model size leads to significant performance gains. This suggests that scaling up the model can offset the negative impact of using a deeper base model, whose total parameter count is typically larger, on the proportion of parameters allocated to the draft model.

Conclusion

In this work, we first identified the dilemma of SD under modern inference services: batch processing compresses available extra computational resources, thereby limiting the draft size. Furthermore, the large number of position-dependent parameters in current draft model architectures hinders their ability to scale effectively, making it challenging to increase parameter size to improve prediction accuracy. Then we proposed a novel SD method for LLMs, termed SpecFormer, which leverages two types of attention mechanisms operating along different dimensions, one unidirectional and one bidirectional. This design enables efficient parallel generation of future tokens while extracting information from the full context, resulting in a more capable draft model. Consequently, our approach maintains high prediction accuracy under a limited draft token budget. We further conduct experiments across varying batch sizes, demonstrating that our method sustains comparable performance as batch size increases. Lastly, evaluations on models of different scales confirm the general applicability of our approach across a broad range of LLM configurations.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 62306216), the Fundamental Research Funds for the Central Universities (No.2042025kf0026), the Technology Innovation Program of Hubei Province (Grant No. 2024BAB043), and the Xiaomi Open-Competition Research Program.

References

Bachmann, G.; Anagnostidis, S.; Pumarola, A.; Georgopoulos, M.; Sanakoyeu, A.; and et al. 2025. Judge Decoding: Faster Speculative Sampling Requires Going Beyond Model Alignment. In *The Thirteenth International Conference on Learning Representations*.

Cai, T.; Li, Y.; Geng, Z.; Peng, H.; Lee, J. D.; and et al. 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.

Chen, J.; Li, Q.; Li, Z.; Qi, B.; Guoming, L.; Ai, H.; Zhao, H.; and Wang, P. 2025. Faster In-Context Learning for LLMs via N-Gram Trie Speculative Decoding. In Christodoulopoulos, C.; Chakraborty, T.; Rose, C.; and Peng, V., eds., *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 18051–18062. Suzhou, China: Association for Computational Linguistics. ISBN 979-8-89176-332-6.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; de Oliveira Pinto, H. P.; and et al. 2021. Evaluating Large Language Models Trained on Code. *CoRR*, abs/2107.03374.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; and et al. 2021. Training Verifiers to Solve Math Word Problems. *CoRR*, abs/2110.14168.

Dao, T. 2024. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Dao, T.; Fu, D.; Ermon, S.; Rudra, A.; and Ré, C. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *Advances in Neural Information Processing Systems*, volume 35, 16344–16359. Curran Associates, Inc.

DeepSeek-AI. 2024. DeepSeek-V3 Technical Report. *CoRR*, abs/2412.19437.

Ding, N.; Chen, Y.; Xu, B.; Qin, Y.; Zheng, Z.; and et al. 2023. Enhancing Chat Language Models by Scaling High-quality Instructional Conversations. arXiv:2305.14233.

Gao, X.; Xie, W.; Xiang, Y.; and Ji, F. 2025. Falcon: Faster and Parallel Inference of Large Language Models Through Enhanced Semi-Autoregressive Drafting and Custom-Designed Decoding Tree. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(22): 23933–23941.

Gloeckle, F.; Idrissi, B. Y.; Rozière, B.; Lopez-Paz, D.; and Synnaeve, G. 2024. Better & Faster Large Language Models via Multi-token Prediction. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Guo, J.; Li, Z.; Wu, J.; Wang, Q.; Li, Y.; Zhang, L.; Zhao, H.; and Yang, Y. 2025. ToM: Leveraging Tree-oriented MapReduce for Long-Context Reasoning in Large Language Models. In Christodoulopoulos, C.; Chakraborty, T.; Rose, C.; and Peng, V., eds., *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 17804–17823. Suzhou, China: Association for Computational Linguistics. ISBN 979-8-89176-332-6.

Hu, Y.; Liu, Z.; Dong, Z.; Peng, T.; McDanel, B.; and Zhang, S. Q. 2025. Speculative Decoding and Beyond: An In-Depth Survey of Techniques. *CoRR*, abs/2502.19732.

Huang, K.; Guo, X.; and Wang, M. 2024. SpecDec++: Boosting Speculative Decoding via Adaptive Candidate Lengths. *CoRR*, abs/2405.19715.

Kamath, A.; Ferret, J.; Pathak, S.; Vieillard, N.; Merhej, R.; and et al. 2025. Gemma 3 Technical Report. *CoRR*, abs/2503.19786.

Kim, S.; Mangalam, K.; Moon, S.; Malik, J.; Mahoney, M. W.; and et al. 2023. Speculative Decoding with Big Little Decoder. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems*, volume 36, 39236–39256. Curran Associates, Inc.

Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; and et al. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, 611–626. New York, NY, USA: Association for Computing Machinery. ISBN 9798400702297.

Lasby, M.; Sinnadurai, N.; Manohararajah, V.; Lie, S.; and Thangarasa, V. 2025. SD^2: Self-Distilled Sparse Drafters. arXiv:2504.08838.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2025a. EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test. *CoRR*, abs/2503.01840.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2025b. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. arXiv:2401.15077.

OpenAI. 2023. GPT-4 Technical Report. *CoRR*, abs/2303.08774.

See, A.; Liu, P. J.; and Manning, C. D. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In Barzilay, R.; and Kan, M., eds., *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, 1073–1083. Association for Computational Linguistics.

Shi, L.; Zhang, H.; Yao, Y.; Li, Z.; and Zhao, H. 2024. Keep the Cost Down: A Review on Methods to Optimize LLM' s KV-Cache Consumption. *CoRR*, abs/2407.18003.

Stern, M.; Shazeer, N.; and Uszkoreit, J. 2018. Blockwise Parallel Decoding for Deep Autoregressive Models. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-

Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Tang, Z.; Luohe, S.; Li, Z.; Qi, B.; Guoming, L.; Zhang, L.; and Wang, P. 2025. SpindleKV: A Novel KV Cache Reduction Method Balancing Both Shallow and Deep Layers. In Che, W.; Nabende, J.; Shutova, E.; and Pilehvar, M. T., eds., *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 28428–28442. Vienna, Austria: Association for Computational Linguistics. ISBN 979-8-89176-251-0.

Taori, R.; Gulrajani, I.; Zhang, T.; Dubois, Y.; Li, X.; Guestrin, C.; Liang, P.; and Hashimoto, T. B. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

Tillet, P.; Kung, H.; and Cox, D. D. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In Mattson, T.; Muzahid, A.; and Solar-Lezama, A., eds., *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2019, Phoenix, AZ, USA, June 22, 2019*, 10–19. ACM.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; and et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *CoRR*, abs/2307.09288.

Wang, Z.; Li, S.; Zhou, Y.; Li, X.; Gu, R.; and et al. 2024. Revisiting SLO and Goodput Metrics in LLM Serving. *CoRR*, abs/2410.14257.

Williams, S.; Waterman, A.; and Patterson, D. A. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4): 65–76.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; and et al. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *CoRR*, abs/1910.03771.

Xia, H.; Ge, T.; Wang, P.; Chen, S.; Wei, F.; and Sui, Z. 2023. Speculative Decoding: Exploiting Speculative Execution for Accelerating Seq2seq Generation. In Bouamor, H.; Pino, J.; and Bali, K., eds., *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, 3909–3925. Association for Computational Linguistics.

Xia, H.; Yang, Z.; Dong, Q.; Wang, P.; Li, Y.; and et al. 2024. Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Findings of the Association for Computational Linguistics ACL 2024*, 7655–7671. Bangkok, Thailand and virtual meeting: Association for Computational Linguistics.

Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; and et al. 2024. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*.

Yang, H.; Yao, Y.; Li, Z.; Qi, B.; Guoming, L.; and Zhao, H. 2025. XQuant: Achieving Ultra-Low Bit KV Cache Quantization with Cross-Layer Compression. In Christodoulopoulos, C.; Chakraborty, T.; Rose, C.; and Peng, V., eds., *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 9796–9811. Suzhou, China: Association for Computational Linguistics. ISBN 979-8-89176-332-6.

Yu, G.-I.; Jeong, J. S.; Kim, G.-W.; Kim, S.; and Chun, B.-G. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 521–538. Carlsbad, CA: USENIX Association. ISBN 978-1-939133-28-1.

Zhang, L.; Bao, C.; and Ma, K. 2022. Self-Distillation: Towards Efficient and Compact Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8): 4388–4403.

Zhang, L.; Wang, X.; Huang, Y.; and Xu, R. 2025. Learning Harmonized Representations for Speculative Sampling. In *The Thirteenth International Conference on Learning Representations*.

Zhao, Y.; Li, Z.; Zhao, H.; Qi, B.; and Guoming, L. 2025a. DAC: A Dynamic Attention-aware Approach for Task-Agnostic Prompt Compression. In Che, W.; Nabende, J.; Shutova, E.; and Pilehvar, M. T., eds., *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 19395–19407. Vienna, Austria: Association for Computational Linguistics. ISBN 979-8-89176-251-0.

Zhao, Y.; Peng, Y.; Nguyen, C.; Li, Z.; Wang, X.; Zhao, H.; and Fu, X. 2025b. SmallKV: Small Model Assisted Compensation of KV Cache Compression for Efficient LLM Inference. *CoRR*, abs/2508.02751.

Zheng, L.; Chiang, W.; Sheng, Y.; Zhuang, S.; and et al. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Zhong, Y.; Liu, S.; Chen, J.; Hu, J.; Zhu, Y.; and et al. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 193–210. Santa Clara, CA: USENIX Association. ISBN 978-1-939133-40-3.

Zhou, Y.; Lyu, K.; Rawat, A. S.; Menon, A. K.; Rostamizadeh, A.; Kumar, S.; and et al. 2024. DistillSpec: Improving Speculative Decoding via Knowledge Distillation. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.