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

Anonymous ACL submission

Abstract

Speculative decoding accelerates LLM inference by utilizing otherwise idle computational resources during memory-to-chip data transfer. Current approaches typically rely on smaller causal models to autoregressively sample draft tokens, often enhanced with prefix trees to explore multiple potential drafts. However, these methods face significant performance degradation as batch size increases, due to reduced surplus computational capacity for speculative decoding. To address this limitation, 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.

1 Introduction

Large language models (LLMs) based on Transformer (Vaswani et al., 2017) 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 (Chowdhery et al., 2023). 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).

However, during inference, generating one token at a time results in low arithmetic intensity (AI, Williams et al., 2009). Each model parameter, once loaded from memory into the chip, typically contributes to only two operations, a multiplication and an addition. In contrast, modern hardware can perform dozens to hundreds of operations in the time it takes to move a single datum from memory, 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 arithmetic intensity. Its core idea is to generate multiple tokens per forward pass of the large model. The process consists of three main steps (Stern et al., 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 contextual information.

Since multiple tokens are generated in one forward pass, speculative decoding is also referred to as multi-token prediction (MTP). **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 SD a purely acceleration-oriented technique, ensuring strict mathematical equivalence with the original model outputs. While relaxing accuracy requirements can potentially yield further speedup, the resulting performance degradation is often difficult to quantify or control, and may unnecessarily complicate the problem. Moreover, empirical observations suggest that algorithms performing well under the lossless SD setting tend to also exhibit strong performance under lossy conditions.

A key observation about this paradigm is that SD does not reduce the total amount of computation, in fact, it often increases it significantly. 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, mentioned earlier, 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). The former typically employs a smaller auxiliary causal model: during generation, the small model accesses partial states from the large model and uses autoregressive decoding to rapidly generate multiple subsequent tokens—benefiting from its smaller size and lower computational cost. The latter, in contrast, utilizes non-autoregressive techniques by storing multiple sets of position-specific parameters. These parameters are used to directly generate draft tokens from the large model's internal states, with each parameter set responsible for predicting a draft token at a specific future position. In both paradigms, these models are often combined with prefix trees, where instead of sampling a single best draft sequence, multiple suboptimal candidates are sampled in parallel (Li et al., 2024). These are merged into a prefix tree and collectively verified by the large model. However, a key challenge arises in batched inference settings, where the residual compute capacity available for speculative decoding is significantly reduced. This severely limits the size of the prefix tree, often degenerating it into a single linear sequence, thereby limiting the predictive accuracy that could otherwise be gained from broader exploration. Moreover, the auxiliary model's parameters are either position-dependent or autoregressive, require repeated memory access for sequential decoding. In both cases, scaling up the auxiliary model to improve prediction quality becomes difficult, as it incurs substantial additional cost or inefficiency.

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.

We evaluate our proposed method on models of approximately 4B, 7B, and 14B parameters (Yang et al., 2024a,b), 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.

2 Background and Related Works

2.1 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.

2.2 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 et al., 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-step LHS, and recursively feeds it into itself. The resulting LHS representations 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 et al., 2024), etc.(Gao 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.

3 Methods

3.1 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 different deployment scenarios. We contend that a more meaningful and necessary 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.

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

3.2 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.

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

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

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

Finally, we define Equation 7 to simplify the description of pre-norm residual connected units (He et al., 2016).

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

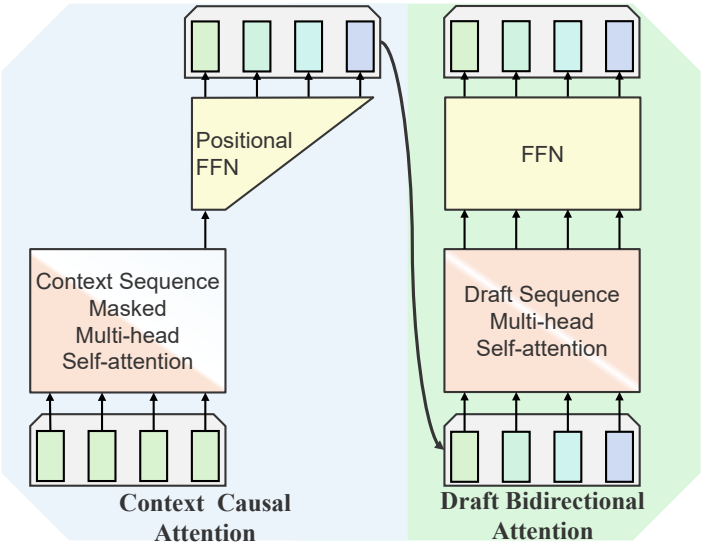

Figure 1: An overview of proposed SpecFormer speculative decoding method.

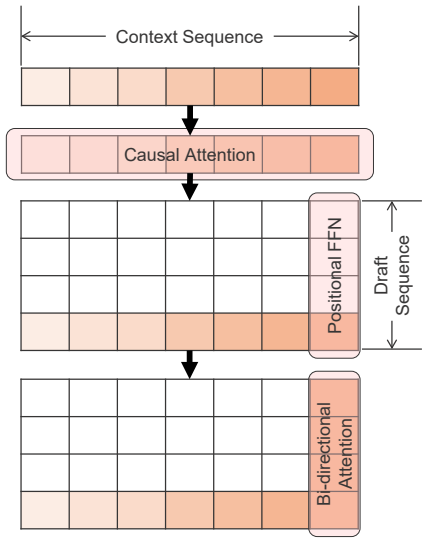

Figure 2: A depiction of uni and bi-directional attention.

3.3 SpecFormer

Our proposed SpecFormer architecture comprises a Context Causal Attention layer and a Draft Bi-directional Attention layer, depicted in Figure 1, incorporating both unidirectional and bidirectional attention along two dimensions, as shown in Figure 2.

3.3.1 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, Zhang and Sennrich, 2019), 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 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.

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

The Positional FFN is a simple 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 significantly 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 of the spectrum. 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.

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

3.3.2 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, Shazeer, 2020) feed-forward 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.

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

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.

3.4 Implementation Improvements

3.4.1 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 et al., 2019) to implement the Grouped RMS Norm operation more efficiently.

3.4.2 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 (Dubey et al., 2024), making the full softmax projection 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.

4 Experiment

4.1 Setups

4.1.1 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 et al., 2022; Lasby et al., 2025) first. Specifically, we retained only the question (prompt) 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, rather than being influenced by another teacher model. Our experiments demonstrate that this adjustment leads to significant performance improvements.

4.1.2 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.

4.1.3 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 et al., 2017; Yu et al., 2021). 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 before and after acceleration.

4.1.4 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 decoding code based on the *HuggingFace Transformers* (Wolf et al., 2019) library. We conducted tests under various batch sizes, and

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.

report the theoretical speedup, efficiency factor κ, and actual speed gains. Our detailed training hyperparameters is given in Appendix A.

4.2 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.

4.3 Special Case Study

4.3.1 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.

4.3.2 Base LLM Size

To investigate the performance gains of our architecture under speculative decoding across differ-

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.

ent 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 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.

5 Conclusion

We first analyze that the batch execution environment imposes constraints on the effectiveness of speculative decoding by decreasing the idle computational resources. Then we proposed a novel speculative decoding 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 whole 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.

Limitations

Our method has several limitations that highlight possible directions for future work. First, it requires training, even though we only train the draft-module, which imposes relatively modest demands in terms of compute and supervision, the inclusion of a self-distillation stage still entails a non-trivial number of GPU-hours. Fully training-free approaches may represent a promising avenue for further research.

Moreover, our method, as with any non-autoregressive decoding strategy, faces inherent challenges when integrated with prefix tree structures, where autoregressive methods currently hold a clear advantage. Developing more effective and efficient mechanisms to couple non-autoregressive predictors with prefix-based verification remains an open and valuable research problem.

Ethics Statement

This work does not involve the collection or use of any personally identifiable data, human subjects, or sensitive information. All experiments are conducted using publicly available datasets and open-source models. We adhere to the principles of responsible AI research, including transparency, reproducibility, and fairness. Any use of large language models complies with the respective licensing terms. Our proposed methods are intended for research purposes only and should be deployed with care to avoid misuse or unintended consequences.

References

Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas K Kohler. 2025. Judge decoding: Faster speculative sampling requires going beyond model alignment. In *The Thirteenth International Conference on Learning Representations*.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé 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, and 39 others. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, and 48 others. 2023. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Tri Dao. 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.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359. Curran Associates, Inc.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 81 others. 2024. Deepseek-v3 technical report. *CoRR*, abs/2412.19437.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. Enhancing chat language models by scaling high-quality instructional conversations. *Preprint*, arXiv:2305.14233.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, and 82 others. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Xiangxiang Gao, Weisheng Xie, Yiwei Xiang, and Feng Ji. 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.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. 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.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yunhai Hu, Zining Liu, Zhenyuan Dong, Tianfan Peng, Bradley McDanel, and Sai Qian Zhang. 2025. Speculative decoding and beyond: An in-depth survey of techniques. *CoRR*, abs/2502.19732.

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

Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean-Bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, and 79 others. 2025. Gemma 3 technical report. *CoRR*, abs/2503.19786.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. 2023. Speculative decoding with big little decoder. In *Advances in Neural Information Processing Systems*, volume 36, pages 39236–39256. Curran Associates, Inc.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA. Association for Computing Machinery.

Mike Lasby, Nish Sinnadurai, Valavan Manohararajah, Sean Lie, and Vithursan Thangarasa. 2025. Sd^2: Self-distilled sparse drafters. *Preprint*, arXiv:2504.08838.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. EAGLE-2: faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 7421–7432. Association for Computational Linguistics.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025a. EAGLE-3: scaling up inference acceleration of large language models via training-time test. *CoRR*, abs/2503.01840.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025b. Eagle: Speculative sampling requires rethinking feature uncertainty. *Preprint*, arXiv:2401.15077.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1073–1083. Association for Computational Linguistics.

Noam Shazeer. 2020. GLU variants improve transformer. *CoRR*, abs/2002.05202.

Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. 2024. Keep the cost down: A review on methods to optimize llm' s kv-cache consumption. *CoRR*, abs/2407.18003.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Philippe Tillet, Hsiang-Tsung Kung, and David D. Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2019, Phoenix, AZ, USA, June 22, 2019*, pages 10–19. ACM.

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, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Zhibin Wang, Shipeng Li, Yuhang Zhou, Xue Li, Rong Gu, Nguyen Cam-Tu, Chen Tian, and Sheng Zhong. 2024. Revisiting SLO and goodput metrics in LLM serving. *CoRR*, abs/2410.14257.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 3909–3925. Association for Computational Linguistics.

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

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024b. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Dian Yu, Kai Sun, Dong Yu, and Claire Cardie. 2021. Self-teaching machines to read and comprehend with large-scale multi-subject question-answering data. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pages 56–68. Association for Computational Linguistics.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, Carlsbad, CA. USENIX Association.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. 2025. Learning harmonized representations for speculative sampling. In *The Thirteenth International Conference on Learning Representations*.

Linfeng Zhang, Chenglong Bao, and Kaisheng Ma. 2022. Self-distillation: Towards efficient and compact neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4388–4403.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *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*.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 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)*, pages 193–210, Santa Clara, CA. USENIX Association.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 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.

A Training Hyperpameters

Reference Table 4. Training takes about 72, 50 and 40 GPU-hours for 14B, 8B and 4B variants, with about 16, 12, 10 GPU-hours for data preparing in self-distillation.

Hyperparameter	Value
Batch Size	2
Grad. Acc.	8
Max Seq. Len.	4096
Num Epochs	2
Total Steps	$463,888$
Max Learning Rate	5e-4 (4B Model) 3e-4 (8B Model) 2e-4 (14B Model)
Min Learning Rate	1e-5 (4B Model) 1e-5 (8B Model) 1e-5 (14B Model)
Warm Up	5% Total steps
Scheduler	Cosine Annealing
Optimizer	AdamW
Adam ϵ	2e-4
Adam βs	$(0.9, 0.999)$
Weight Decay	0.01

Table 4: Hyperparameters used for training.

B Hardware Detail

Please reference Table 5.

Item	Value
CPU	24 * Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz
GPU	NVIDIA A800 PCIe 80 GB
RAM	212GB DDR4-2667

Table 5: Hardware used.