Speculative Streaming: Efficient and Scalable Speculative Decoding with Multi-Stream Attention

Anonymous ACL submission

Abstract

Speculative decoding is a prominent technique for accelerating LLM inference by leveraging an auxiliary draft model, but its effectiveness is limited by the autoregressive nature of draft generation, where acceptance rates depend on the draft model's size. Scaling the draft model improves acceptance but also increases speculation latency, limiting overall speedup. Furthermore, fine-tuning both the draft and target models is often necessary to achieve high acceptance rates, adding complexity to inference systems as the number of downstream tasks grows. Single-model approaches like Medusa generate speculative tokens non-autoregressively but lack token dependencies, limiting effectiveness. Alternatives like Hydra and Eagle incorporate token dependencies but rely on dedicated heads, making speculation independent of the base model and limiting the extent to which stronger base models can improve speculation.

We introduce a novel speculative decoding method that integrates speculative draft generation directly within the target model using multi-stream attention. This improves acceptance rates by introducing interdependencies between speculative tokens while ensuring non-autoregressive draft generation with minimal overhead. As target models scale in size and quality, speculative generation improves naturally with our method, unlike prior approaches. Furthermore, our approach is both parameter- and FLOP-efficient, requiring over 1000× fewer additional parameters than Medusa, making it highly suitable for resource-constrained devices. We design our method to operate in two modes: (1) Lossless mode, a plug-and-play method that preserves the output of any pre-trained model; and (2) Shared mode, optimizing both speedup and downstream output quality. We demonstrate a 2–3.5× speedup across diverse tasks, including summarization, translation, question answering, mathematical reasoning, SQL generation, and retrieval-augmented generation (RAG).

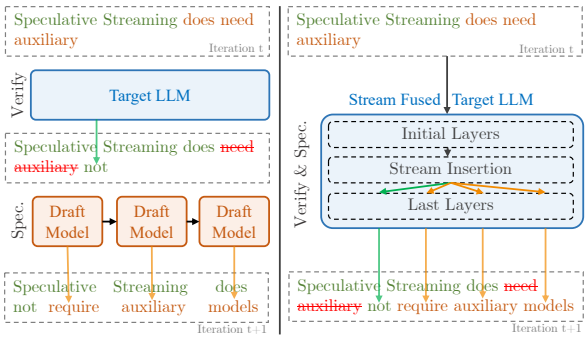


Figure 1: (a) Speculative Decoding requires a well-aligned draft model that generates speculation autoregressively. (b) Speculative Streaming significantly simplifies the system by performing speculation and verification concurrently, within a single stream-fused model.

1 Introduction

Large transformers have become the cornerstone of modern language modeling. The quality of these models improves as they scale (Kaplan et al., 2020), leading to the introduction of the state-of-the-art multi-billion parameter models (Brown et al., 2020; Thoppilan et al., 2022; Chowdhery et al., 2023; Touvron et al., 2023a). While these models are effective for token generation, they incur a high inference cost as the model and its transient states need to be loaded into computing memory for each newly generated token. This poses a challenge to the deployment of large autoregressive transformers, particularly for user-facing applications with stringent latency requirements.

Given the memory-bound nature of large language model (LLM) inference, recent work (Leviathan et al., 2023; Chen et al., 2023) has proposed Speculative Decoding as an effective technique to accelerate decoding based on concepts borrowed from speculative computation (Burton, 1985). The core idea is to speculate multiple candidate future tokens first and then verify them

in parallel using a two-model paradigm as shown in Figure 1a: a small auxiliary "draft" model for candidate speculation and a large "target" model for verification (Leviathan et al., 2023; Chen et al., 2023). However, the effectiveness of speculative decoding is constrained by the autoregressive nature of draft generation, where the draft model's size directly impacts acceptance rates; scaling the draft model improves quality but also increases speculation latency, limiting the overall speedup. It is also resource-inefficient, requiring both models to be hosted in memory during token prediction.

In this paper, we propose *Speculative Streaming*, a single-model speculative decoding approach that unifies speculation and verification, obviating the need for a separate draft model as shown in Figure 1b. This is accomplished by incorporating multi-stream attention into the target model to perform n-gram prediction, which serves as future candidate speculation. As a result, a forward pass can verify the previously generated tokens while simultaneously speculating on the future tokens. Speculative Streaming (SS) improves acceptance rates by introducing interdependencies among speculative tokens while ensuring non-autoregressive draft generation with minimal overhead. Unlike previous approaches, where the quality of speculative draft is independent of target model size, multi-stream attention in our method ensures that speculative generation quality improves naturally as the target model scales in size and quality. The key advantages of Speculative Streaming are as follows:

- Achieves substantial decoding speedups through a unified fine-tuning process that integrates multi-stream attention, preserving exact output in lossless mode while enhancing output quality in shared mode.
- Improves speculative generation quality as model size scales, leveraging the richer representations of larger pre-trained models to enhance both speculation accuracy and verification efficiency.
- Minimizes resource overhead, requiring fewer additional parameters and FLOPs compared to state-of-the-art speculative decoding methods, while still outperforming them in speedup gains.
- Simplifies deployment by eliminating the need for an auxiliary draft model, avoiding the complexity of model alignment, switching, and management during inference, as required by approaches such as (Leviathan et al., 2023).

2 Method

Our goal is to develop an end-to-end trainable, single-model framework that integrates speculative residual streams, aligning residual transformations with future tokens to enable high-acceptance, non-autoregressive speculation. We also aim to address greedy decoding limitations by incorporating plausible future residual states, enabling more informed token selection through future token planning. To achieve these objectives, we introduce the following key components: (a) Speculative stream design and initialization as described in Section 2.1 (b) Parallel speculation and verification as described in Section 2.2 (c) Parallel tree draft pruning, described in Section 2.3 and (d) Training objective as described in Section 2.4.

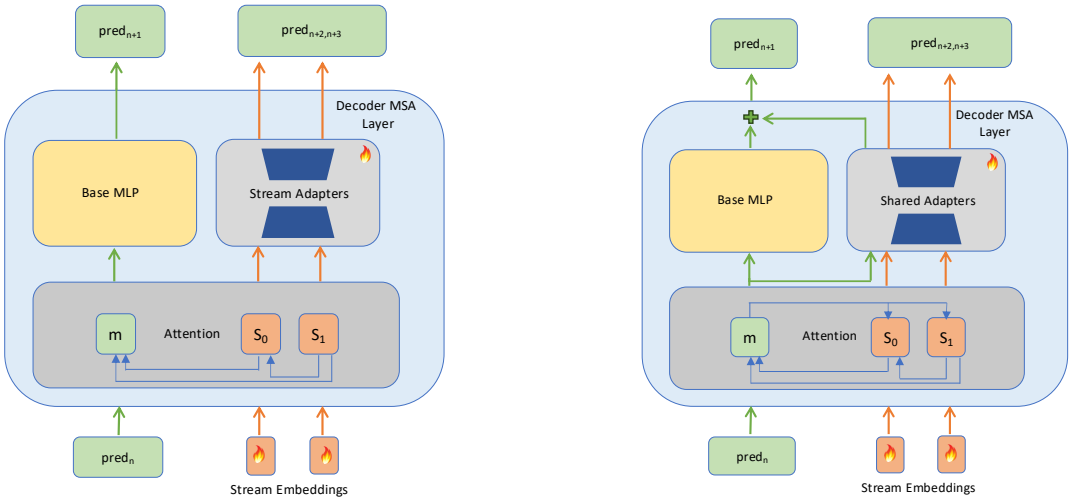
2.1 Streams Design and Initialization

Parameter efficient supervised fine-tuning (Hu et al., 2022) of decoder-only pre-trained language models involves training low-rank adapters to predict next target token y_t given context tokens $(x_1....x_m)$ and previous target tokens $(y_1..y_{<t})$ on downstream applications. At the heart of this process lies the Multi-Head Attention (MHA) mechanism (Vaswani et al., 2017) operating on the residual stream, which can be formally described as:

$$M_t^{k+1} = \text{MHA}(M_t^k, M_{\leq t}^k, M_{\leq t}^k) \qquad (1)$$

where M_t^k denotes base residual stream at time step t and layer k and $MHA(H, H, H)$ denotes attention between query HW^Q, key HW^K and value HW^V as described in (Vaswani et al., 2017). Building upon this framework, we introduce *speculative residual streams*, which attend to the base (main) residual stream via a novel *multi-stream attention (MSA)* mechanism. Each speculative stream is designed to generate future tokens with minimal latency overhead in memory-bound decoding scenarios. Specifically, the added speculative streams predict $p(y_{t+j} \mid y_{<t}, x)$ for $1 \leq j \leq \gamma$, where γ denotes the number of speculative steps, while the base stream continues to predict $p(y_t \mid y_{<t}, x)$.

To seamlessly integrate speculative streams into pre-trained models, we propose a *lossless mode* for scenarios that require keeping the original output distribution of the target model untouched (see Figure 2a). In this mode, the attention mechanism of the base residual stream adheres to the standard MHA formulation (Vaswani et al., 2017). However, each speculative stream j at time step t attends to



(a) Lossless Mode

(b) Shared Mode

Figure 2: Shared and Lossless Modes of Speculative Streaming. In the lossless mode (a), speculative streams attend to the main stream, whereas in the shared mode (b), attention is bidirectional. In the lossless mode, the base model remains frozen while stream embeddings and stream adapters in the MSA decoder layers are trained to predict speculative tokens. In the shared mode, we insert adapters into the base model decoder layers and train them on an n-gram prediction objective. Notably, the adapter parameters in MSA layers are shared to influence the residual transformation of both main and speculative streams.

the prior hidden states of both the base and speculative streams:

$$S_{tj}^{k+1} = \text{MHA}(S_{tj}^{k}, M_{\leq t}^{k} \oplus S_{t(\leq j)}^{k}, M_{\leq t}^{k} \oplus S_{t(\leq j)}^{k}), \quad (2)$$

where S_{tj}^{k} denotes the speculative stream j at layer k and time t, $M_{\leq t}^{k}$ represents the base residual stream up to t and $\oplus$ represents concatenation. At the final transformer layer N, the hidden state M_t^N is used to predict y_t, while each speculative stream's terminal state S_{tj}^{N} predicts y_{t+j}. We refer to decoder layers implementing the standard MHA as *MHA layers*, whereas those incorporating the formulation above as *MSA layers*.

While effective in accelerating decoding, MSA in lossless mode does not modify the base model's objective of greedily generating the next token. To enable proactive token planning and reduce overfitting to local correlations during token generation (Yang et al., 2019; Qi et al., 2020), we propose a *shared mode* (see Figure 2b). In this mode, the training objective is extended from next-token prediction to *n-gram prediction*. This is achieved by allowing the base stream to attend to speculative streams, thereby enabling it to refine its residual transformation using anticipated future states:

$$M_t^{k+1} = \text{MHA}(M_t^{k}, M_{\leq t}^{k} \oplus S_{t1\ldots\gamma}^{k}, M_{\leq t}^{k} \oplus S_{t1\ldots\gamma}^{k}). \quad (3)$$

Here, $S_{t1\ldots\gamma}^{k}$ represents all speculative streams at layer k and time t. By integrating future residual states during training, this shared mode aligns the base stream with speculative planning, promoting both efficiency and robustness in token generation.

The attention mechanism of speculative streams remains consistent across both shared and lossless modes (see Equation (2)). Key/value projections of the main stream's residual states are cached during inference to avoid re-computation, whereas we design speculative stream attention specifically to avoid storing additional key/value projections associated with individual streams. This is because speculative streams are trained to learn contextual features using the main stream's key/value context allowing us to not introduce additional caching overhead and operate within memory bounds of resource-constrained devices.

In lossless mode, main stream passes through frozen base MLP layer and speculative streams pass through the parallel stream adapters as shown in Figure 2a. In contrast, shared mode fine-tunes the base model for n-gram prediction via shared adapters. Within MHA decoder layers, these adapters modulate the residual transformation of the main stream, whereas in MSA layers, they influence the residual transformations of both the main and speculative streams, as illustrated in Figure 2b.

We initialize hidden states of speculative streams

at layer $N - N_s$ instead of initializing them from the embedding layer, where N_s denotes the number of MSA layers. Specifically, stream j at time t is initialized at layer $N - N_s$ as

$$S_{tj}^{N-N_s} = f_\eta(M_t^{N-N_s}) + P_j^{N-N_s} \quad (4)$$

where P_j is a stream identifier embedding that embeds a sense of relative position into streams and distinguishes the computation from main stream. f_η is a linear transformation of rank η to transform main stream hidden states into speculative stream hidden states. This initialization helps to reduce computation per forward pass by decreasing the speculative FLOPs contribution by $(N - N_s)/N$. In terms of forward pass latency, FLOPs do not contribute significantly when the model is memory bound, however, as we describe in Section 2.2, we sample additional tokens to shift the model into a compute-bound regime, therefore FLOPs reduction becomes crucial.

2.2 Parallel Speculation and Verification

In standard draft-target speculative decoding (Leviathan et al., 2023), speculation and verification processes happen sequentially. Speculative Streaming makes this process efficient by parallelizing speculation and verification. In each forward pass, the draft from the previous step is verified and a new draft is generated as shown in Figure 3. For instance, in step s, if draft tokens $(\tilde{y}_1 \dots \tilde{y}_\delta)$ are accepted where $0 < \delta \leq \gamma$, main stream M_δ is used to issue a correction token, and logits from speculative streams $S_{\delta(1\dots\gamma)}$ are used to generate draft for step $s + 1$.

Instead of using a linear sequence of speculated tokens for verification, we sample a tree of tokens from main and speculative streams, where each path in the tree represents one possible verification candidate. Tree drafting enables accepting the longest matching candidate sequence and more tokens can be advanced during each forward pass. To create a tree draft, instead of sampling 1 token from logits of speculative streams, $(z_1 \dots z_\gamma)$, we sample top k tokens and form a tree of sampled tokens as shown in Figure 3, such that tokens sampled from stream n are predecessors of tokens sampled from stream $n + 1$. We process a tree draft of speculative tokens in one forward pass by creating an additive attention mask (Vaswani et al., 2017), such that each node in the tree attends to its predecessor. Attention mask between k^{th} token sampled from logits of stream j, $\tilde{y}_{jk}$ and the m^{th} token sampled from logits of stream n, $\tilde{y}_{nm}$ is defined as:

$$a_{\tilde{y}_{jk}\tilde{y}_{nm}} = \begin{cases} 0 & \text{if j = n+1,} \\ -\infty & \text{otherwise} \end{cases} \quad (5)$$

Refer to Figure 15 for more details.

2.3 Parallel Tree Pruning

A key challenge in constructing speculative tree drafts is the combinatorial explosion of candidate paths: sampling k tokens from each of γ streams yields a tree draft of size $1 + \sum_{g=1}^{\gamma} k^g$. To enable parallel draft generation in a single forward pass, each draft token is batched with γ speculative streams in MSA layers, resulting in a total batch size of $(1+\gamma)(1+\sum_{g=1}^{\gamma} k^g)$. As batch size grows, target model inference becomes compute-bound, diminishing the latency gains from wider sampling.

To address this, we introduce a parallel tree draft pruning layer that eliminates low-probability tokens based on transition likelihoods between parent and immediate child tokens. While prior methods prune speculative trees based on draft model confidence (Anonymous, 2024), they suffer from overconfidence, where high-confidence paths may ultimately be rejected by the target model, and underconfidence, where low-confidence paths are prematurely pruned despite being acceptable. In contrast, we prune using early-exit confidence from the target model itself. Specifically, hidden states M^l at layer l are projected via a low-rank transformation o_θ, and passed through the language modeling head H to yield early-exit logits $\tilde{z} = H(o_\theta(M^l))$. The transition score $\tilde{z}_{pc}$ approximates the probability of a child token c given its parent p. These early acceptances align closely with final verification outcomes, as residual change tends to decrease across deeper layers (Bhendawade et al., 2025), making early-layer agreement a reliable proxy.

The pruning layer can be flexibly inserted at any depth, balancing pruning accuracy and latency: earlier layers reduce compute but risk false rejections; deeper layers improve precision at higher cost. In all experiments in Section 3, we insert the pruning layer immediately before speculative stream insertion. Please refer to Appendix G.1 for additional implementation details.

2.4 Training

In the shared mode (see Figure 2b), our instruction tuning procedure entails training the adapters and

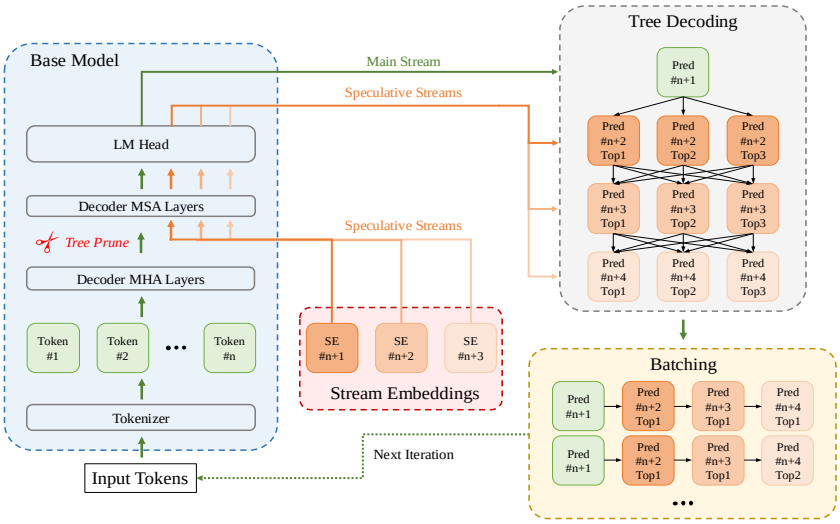


Figure 3: Top level architecture: We replace top N_s multi-head attention (MHA) layers of the base model with multi-stream attention (MSA) layers as described in (2). Speculative streams are initialized using hidden states of layer $N - N_s$ and stream identifier embeddings (SE), as described in (4) and used to generate speculative draft in the form of a tree. The speculative tree draft from the previous iteration is batched for verification and pruned using early exit based confidence before stream insertion as discussed in Section 2.3. During each forward pass previous pruned tree draft is verified and a new tree draft is issued using speculative streams as described in Section 2.2.

stream embeddings on both the prediction loss of the next token and γ future tokens. The overall loss function is defined as follows:

$$L_{ss} = -\alpha_0 \left(\sum_{t=1}^{T} \log p_\theta(y_t | y_{<t}, x) \right) - \sum_{j=1}^{\gamma} \alpha_j \left(\sum_{t=1}^{T-j} \log p_\theta(y_{t+j} | y_{<t}, x) \right) \quad (6)$$

where α_0 and α_j are set empirically to normalize losses of the next token and speculative token prediction. In lossless mode, only the stream adapters and embeddings are trained for speculative token prediction, while the base model remains frozen, with α_0 set to 0 (see Figure 2a).

Although training with Speculative Streaming is relatively cheap (see Appendix J), naive training increases batch dimension along sequence length axis by γ in MSA layers, causing attention computation to hit peak memory with larger batches. We employ a segment-based attention method that helps reduce peak memory consumption and increases training throughput significantly by dividing training sample into prompt and multiple completion segments. More details on segment attention can be found in Appendix I. Finally, the tree pruning adapter described in Section 2.3 is trained on next token prediction loss.

3 Experiments

We evaluate our method on diverse tasks from open speculative decoding benchmarks, as well as on a set of applications vital to on-device AI assistants.

Datasets. To evaluate the effectiveness of Speculative Streaming in multi-turn interactive conversations and tasks such as reasoning and coding, we train both shared and lossless variants on the ShareGPT dataset and measure decoding speedup using MT-Bench (Zheng et al., 2023).

To ensure broader generalizability, we integrate the lossless variant of our approach with SpecBench (Xia et al., 2024), a benchmark designed to assess the effectiveness of speculative decoding methods in lossless settings. Additionally, we compare our supervised fine-tuning objective described in Section 2.4 against traditional next-token prediction based fine-tuning across key applications for on-device AI assistants, including Text Summarization using DialogSum dataset (Chen et al., 2021), Structured Query Generation using SqlCreateContext dataset constructed from WikiSQL (Zhong et al., 2017a) and SPIDER (Yu et al., 2018), and Meaning Representation using E2E-NLG dataset (Dušek et al., 2020).

Model Configurations and Baselines. We evaluate our method on open-source models of varying scales, including Llama-2-Chat (7B, 13B) (Touvron et al., 2023b) and Vicuna (7B, 13B, 33B) (Chi-

Table 1: Comparison of wall-time speedup and generation quality scores across Llama and Vicuna models of varying scales on MT-Bench. Results for Medusa, Medusa-2, Hydra, Eagle and LookAhead decoding are taken from their respective papers, with Hydra results corresponding to the best-performing variant, Hydra++. Notably, methods like Eagle and 2-Model SD generate speculative tokens autoregressively, whereas Speculative Streaming produces them in a non-autoregressive (NAR) manner as shown in Table 6. As a result, even though the mean number of accepted tokens of Speculative Streaming on smaller models are slightly lower than Eagle, wall-time speedup is higher due to the absence of autoregressive speculation generation overhead.

Model	Method	Speedup (↑)	Score (↑)
Vicuna-7B	Baseline	1x	6.17
	2-Model SD	1.42x	6.17
	Medusa	2.18x	6.17
	Hydra	2.70x	6.17
	Medusa-2	2.83x	6.18
	Eagle	2.90x	6.17
	SS-Lossless (Ours)	3.08x	6.17
	SS-Shared (Ours)	**3.22x**	**6.21**
Vicuna-13B	Baseline	1x	6.39
	2-Model SD	1.55x	6.39
	Medusa	2.33x	6.39
	Hydra	2.50x	6.39
	Medusa-2	2.83x	6.43
	Eagle	3.07x	6.39
	SS-Lossless (Ours)	3.21x	6.39
	SS-Shared (Ours)	**3.29x**	**6.48**
Vicuna-33B	Baseline	1x	7.12
	2-Model SD	1.59x	7.12
	Medusa	1.98x	7.12
	Hydra	2.53x	7.12
	Medusa-2	2.35x	7.18
	Eagle	2.95x	7.12
	SS-Lossless (Ours)	3.24x	7.12
	SS-Shared (Ours)	**3.35x**	**7.22**
Llama-2-Chat-7B	Baseline	1x	6.27
	2-Model SD	1.39x	6.27
	LookAhead	1.64x	6.27
	Eagle	2.78x	6.27
	SS-Lossless (Ours)	2.93x	6.27
	SS-Shared (Ours)	**3.05x**	**6.29**
Llama-2-Chat-13B	Baseline	1x	6.65
	2-Model SD	1.47x	6.65
	LookAhead	1.51x	6.65
	Eagle	3.03x	6.65
	SS-Lossless (Ours)	3.23x	6.65
	SS-Shared (Ours)	**3.30x**	**6.71**

ang et al., 2023) to demonstrate the scalability of our approach. For application-specific settings, we test our approach on Phi-3-mini-4k-instruct (3.8B) (Abdin et al., 2024), Mistral (7B) (Jiang et al., 2023), and OPT (1.3B, 6.7B) (Zhang et al., 2022).

We compare our approach against draft-target speculative decoding methods (Leviathan et al., 2023; Zhou et al., 2023) as well as single-model speculative decoding frameworks, including Medusa (Cai et al., 2023), LookAhead Decoding (Fu et al., 2023), Hydra (Ankner et al., 2024), Eagle (Li et al., 2024), Prompt Lookup Decoding (PLD) (Saxena, 2023) , REST (He et al., 2024), and SPACE (Yi et al., 2024). For the standard draft-target approach, we use OPT-125M as the draft model for OPT-1.3B and OPT-6.7B target models. For Vicuna and Llama models, we adopt `JackFram/llama-68M` (Miao et al., 2023) as the draft model. All draft models are fine-tuned on the ShareGPT dataset to ensure evaluation fairness.

Metrics. On MT-Bench, we evaluate response quality using GPT-4-graded scores that assess coherence, correctness, and engagement. We conduct our experiments using FastChat (Zheng et al., 2023), which incorporates GPT-4 evaluation. Since SpecBench is primarily designed for lossless speculative decoding methods, we integrate the lossless version of our technique and report the wall-time speedup. In application-specific settings, we evaluate both wall-time speedups and generation quality metrics. For the structured query task, we use Exact Match (EM) accuracy, while for Dialog Summarization and Meaning Representation tasks, we report ROUGE-1 and ROUGE-LSum scores. Finally, to demonstrate the deployment benefits of our approach, we report the parameter and FLOP overhead associated with our method.

Inference. Inference is performed using a batch size of 1 on a single Nvidia A100-80G GPU in float16 using greedy sampling and $T = 0$, reflecting the typical deployment setting for on-device assistants. Please refer to Appendix L for batching impact, Appendix E.3 for ablations on top-k sampling with $T = 1$, Appendix E.5 for quantization impact and Appendix N.1 for more experimental details. We set $N_s = 4$, $\gamma = 4$ and $k = 3$. Please refer to Appendix E for hyperparameter ablations.

3.1 Results

Effectiveness on MT Bench Table 1 presents a comparative evaluation of our method on MT-Bench in terms of speedup and MT-Bench scores. Our experimental results demonstrate that both variants of Speculative Streaming—Lossless and Shared—achieve substantial acceleration across model scales. The Lossless variant yields speedup factors of 2.93–3.24× while preserving the base model's output. The Shared variant achieves slightly higher speedups of 3.05 – 3.35×, with comparable or superior MT-Bench scores, indicating that the adapter parameter sharing strategy further reduces latency while maintaining or improving the generation quality. Both variants consistently

Table 2: Wall-time speedup of lossless Speculative Streaming (SS) across various tasks, evaluated using the comprehensive SpecBench framework. SS consistently outperforms other baselines, with speedups increasing as model size grows, highlighting the scalability of our approach.

Model	Task	EAGLE	Hydra	Medusa	PLD	SPACE	REST	Lookahead	SS-Lossless
Vicuna-7B	Translation	1.99x	1.94x	1.73x	1.04x	1.13x	1.31x	1.14x	**2.32x**
	Summarization	2.23x	1.79x	1.57x	2.43x	1.62x	1.36x	1.19x	**2.51x**
	Question Answering	2.12x	2.03x	1.75x	1.14x	1.49x	1.66x	1.24x	**2.11x**
	Mathematical Reasoning	2.67x	2.49x	2.05x	1.61x	1.47x	1.21x	1.55x	**2.89x**
	Retrieval Augmented Generation	2.04x	1.77x	1.51x	1.71x	1.55x	1.73x	1.09x	**2.53x**
Vicuna-13B	Translation	1.96x	1.90x	1.66x	1.02x	1.13x	1.17x	1.06x	**2.38x**
	Summarization	2.44x	1.93x	1.63x	2.19x	1.68x	1.37x	1.20x	**2.62x**
	Question Answering	2.04x	1.96x	1.63x	1.03x	1.39x	1.53x	1.12x	**2.16x**
	Mathematical Reasoning	2.70x	2.48x	2.00x	1.57x	1.53x	1.19x	1.48x	**3.37x**
	Retrieval Augmented Generation	2.23x	1.92x	1.58x	1.71x	1.67x	1.55x	1.12x	**2.78x**
Vicuna-33B	Translation	2.05x	2.01x	1.73x	1.06x	1.28x	1.27x	1.08x	**2.41x**
	Summarization	2.51x	2.04x	1.64x	2.00x	1.76x	1.45x	1.20x	**2.77x**
	Question Answering	2.17x	2.11x	1.66x	1.07x	1.53x	1.61x	1.16x	**2.23x**
	Mathematical Reasoning	2.99x	2.71x	2.07x	1.55x	1.69x	1.30x	1.54x	**3.45x**
	Retrieval Augmented Generation	2.27x	2.06x	1.62x	1.45x	1.68x	1.61x	1.15x	**2.83x**

outperform all baselines in terms of wall-time speedups. The mean number of accepted tokens of our approach is comparable to those of Eagle, Hydra, and 2-Model SD as shown in Table 6. Notably, unlike these baselines, our method generates tokens in a non-autoregressive (NAR) manner, avoiding the additional overhead associated with autoregressive (AR) speculative token generation, leading to higher speedups. In Appendix N, we conduct a comprehensive analysis comparing AR-based speculation with a draft model to NAR-based speculation using our SS framework. Furthermore, our method achieves significantly higher mean number of accepted tokens compared to Medusa, another approach employing non-autoregressive speculative drafts. In Section 3.2, we provide empirical insights into the superior speedups and generation metrics achieved by our approach. Finally, our method incurs significantly lower memory access and computational overhead (see Figure 8 and Figure 9), underscoring its efficiency, scalability, and deployment advantages.

Generalizability on SpecBench We integrate lossless speculative streaming with SpecBench to assess generalizability across diverse language tasks. Our evaluation focuses on the lossless variant, as SpecBench is specifically designed for benchmarking lossless speculative decoding methods. As shown in Table 2, results demonstrate consistent acceleration across all evaluated tasks and model scales. Notably, Mathematical Reasoning task shows the highest speedup factors, reaching 3.45x for Vicuna-33B, followed by Retrieval Augmented Generation task at 2.83x. These improvements consistently surpass existing methods across all task categories. The performance scales effectively with model size, as evidenced by the progression from Vicuna-7B to Vicuna-33B, where larger models demonstrate enhanced speedups.

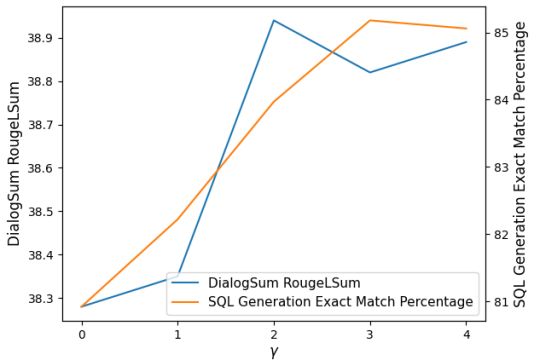


Figure 4: Generation quality of the Phi-3 model when trained to attend to γ ground truth tokens beyond the immediate next token during prediction. Incorporating future ground truth tokens into the attention mechanism leads to substantial improvements in generation quality.

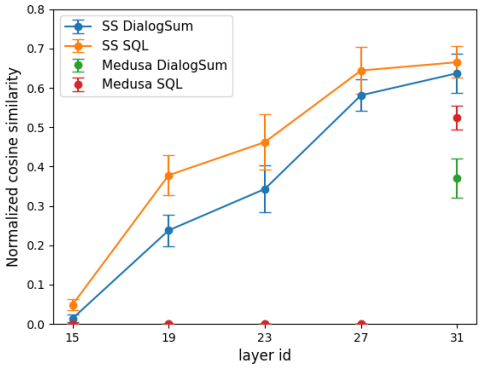


Figure 5: Cosine similarity between speculative residual states and residual state of ground truth tokens with Speculative Streaming and Medusa. As the streams propagate through the model, their representations become increasingly aligned with the ground-truth tokens in contrast to Medusa.

3.2 Why Does It Work?

To deconstruct the empirical effectiveness of Shared Speculative Streaming (SS), we analyze how its architectural design influences both generation quality and decoding efficiency. We begin by isolating the effect of speculative access to future context on next-token prediction.

Generation Metrics: We designed an experiment where the base model predicts the next token while attending to a set of future γ ground truth tokens beyond the next token. Our hypothesis was that by granting the model access to these future tokens, the attention mechanism would enhance its ability to anticipate and plan for the next token, thus improving generation quality. Specifically, we postulated that:

$$p(y_t = g_t | y_{<t}, y_{t+1..t+\gamma}, x) > p(y_t = g_t | y_{<t}, x) \quad (7)$$

Here, g_t represents the ideal ground truth token that maximizes the generation quality metrics. To validate this hypothesis, we modified the attention mask, allowing the model's residual states to "peek" into future residuals. As shown in Figure 4, this modification led to significant improvements in generation metrics.

While such access to future tokens is not feasible during inference, where future states are unavailable, our approach enables the model to approximate future residual states using speculative streams. As demonstrated in Figure 5, these speculative streams, S_{tj}, progressively align with the true residual states of the next tokens as they propagate through the model layers. Since the shared version of our method allows the primary stream, M_t, to attend not only to the current context up to token y_t but also to the speculative streams S_{tj}, this multi-stream attention mechanism refines the transformations within M_t, aligning them more closely with the context of the upcoming γ tokens. As a result, the model effectively "plans" for future tokens, leading to improvements in generation quality.

Speedup: Approaches such as Medusa generate the hidden states of speculative tokens $y_{(t+1...t+\gamma)}$ by applying a simple context independent transformation to the last hidden state of the current token y_t. However, this method has significant limitations. The absence of attention mechanisms results in lower similarity metrics between the speculative hidden states generated by Medusa and the true hidden states, which are obtained by feeding the actual next token into the model (see Figure 5). In contrast, our proposed technique leverages multi-stream attention, wherein speculative streams are allowed to attend to each other as well as to the main stream. As these streams propagate through the model layers, they more closely approximate the true hidden states of the actual next tokens, resulting in higher similarity, thereby increasing the acceptance rate of the speculated tokens.

For a detailed analysis of the training dynamics underlying these effects, including backward gradient propagation through speculative paths and implicit self-distillation enabled by multi-stream attention, we refer the reader to the extended discussion in Appendix B.

4 Conclusion

We present Speculative Streaming, a novel speculative decoding method leveraging multi-stream attention to approximate future residual states. Through rigorous evaluation across diverse benchmarks, we show that Speculative Streaming consistently delivers 2–3.5× speedup while retaining flexibility to operate in both lossless and shared modes. Moreover, it is highly parameter-efficient, reducing memory access overhead by orders of magnitude relative to prior methods while maintaining or surpassing downstream task performance. Its architectural simplicity, scalability, and effectiveness make it well-suited for deployment in resource-constrained environments, advancing the frontier of speculative decoding techniques.

5 Limitations

Speculative streaming is primarily designed to accelerate decoding on resource-constrained devices with high arithmetic intensity. Although most mainstream neural accelerators (on both edge and servers) follow this trend of having orders of magnitude more compute available relative to their memory bandwidth, in rare cases where available FLOPs/memory bandwidth ratio is significantly low, speculative streaming may not be optimal and optimizations that reduce compute such as early exiting (Schuster et al., 2022), skip decoding (Corro et al., 2023), Mixture of Depths (Raposo et al., 2024) could be a better choice.

Although speculative streaming is considerably more parameter-efficient than other approaches, it does introduce a small number of additional parameters, as detailed in Table 3. To mitigate this overhead, we explored an alternative approach that leverages rotating value projections to differentiate stream computation (see Appendix E.4). However, this method led to some degradation in fine-tuning performance compared to using dedicated stream embeddings, as shown in Figure 12a. Further investigation is needed to refine this technique or develop alternative solutions that completely eliminate the parameter overhead while preserving performance.

References

Marah Abdin et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *Preprint*, arXiv:2404.14219.

Megha Agarwal, Asfandyar Qureshi, Nikhil Sardana, Linden Li, Julian Quevedo, and Daya Khudia. 2023a. Llm inference performance engineering: Best practices.

Rishabh Agarwal, Nino Vieillard, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. 2023b. Gkd: Generalized knowledge distillation for auto-regressive sequence models. *arXiv preprint arXiv:2306.13649*.

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

Anonymous. 2024. Faster speculative decoding via effective draft decoder with pruned candidate tree. arXiv preprint under ACL ARR 2024. ACL ARR 2024 December Submission 676.

Apple. n.d. Use writing tools on your mac. Accessed: 2025-02-09.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Nikhil Bhendawade, Mahyar Najibi, Devang Naik, and Irina Belousova. 2025. M2r2: Mixture of multi-rate residuals for efficient transformer inference. *arXiv preprint arXiv:2502.02040*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

F Warren Burton. 1985. Speculative computation, parallelism, and functional programming. *IEEE Transactions on Computers*, 100(12):1190–1193.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, and Tri Dao. 2023. Medusa: Simple framework for accelerating llm generation with multiple decoding heads. `https://github.com/FasterDecoding/Medusa`.

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

Yulong Chen, Yang Liu, Liang Chen, and Yue Zhang. 2021. DialogSum: A real-life scenario dialogue summarization dataset. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 5062–5074, Online. Association for Computational Linguistics.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, New Orleans, Louisiana. Association for Computational Linguistics.

Luciano Del Corro, Allie Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Awadallah, and Subhabrata Mukherjee. 2023. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference. *arXiv preprint arXiv:2307.02628v1*.

Jiaxi Cui, Zongjian Li, Yang Yan, Bohua Chen, and Li Yuan. 2023. ChatLaw: Open-source legal large language model with integrated external knowledge bases. *arXiv preprint arXiv:2306.16092*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2023. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*.

Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. 2020. Evaluating the State-of-the-Art of End-to-End Natural Language Generation: The E2E NLG Challenge. *Computer Speech & Language*, 59:123–156.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2023. Breaking the sequential dependency of llm inference using lookahead decoding.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543*.

Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, Bowen Zhang, Chen Chen, Chung-Cheng Chiu, David Qiu, Deepak Gopinath, Dian Ang Yap, Dong Yin, Feng Nan, Floris Weers, Guoli Yin, Haoshuo Huang, Jianyu Wang, Jiarui Lu, John Peebles, Ke Ye, Mark Lee, Nan Du, Qibin Chen, Quentin Keunebroek, Sam Wiseman, Syd Evans, Tao Lei, Vivek Rathod, Xiang Kong, Xianzhi Du, Yanghao Li, Yongqiang Wang, Yuan Gao, Zaid Ahmed, Zhaoyang Xu, Zhiyun Lu, Al Rashid, Albin Madappally Jose, Alec Doane, Alfredo Bencomo, Allison Vanderby, Andrew Hansen, Ankur Jain, Anupama Mann, Areeba Kamal, Bugu Wu, Carolina Brum, Charlie Maalouf, Chinguun Erdenebileg, Chris Dulhanty, Dominik Moritz, Doug Kang, Eduardo Jimenez, Evan Ladd, Fangping Shi, Felix Bai, Frank Chu, Fred Hohman, Hadas Kotek, Hannah Gillis Coleman, Jane Li, Jeffrey Bigham, Jeffery Cao, Jeff Lai, Jessica Cheung, Jiulong Shan, Joe Zhou, John Li, Jun Qin, Karanjeet Singh, Karla Vega, Kelvin Zou, Laura Heckman, Lauren Gardiner, Margit Bowler, Maria Cordell, Meng Cao, Nicole Hay, Nilesh Shahdadpuri, Otto Godwin, Pranay Dighe, Pushyami Rachapudi, Ramsey Tantawi, Roman Frigg, Sam Davarnia, Sanskruti Shah, Saptarshi Guha, Sasha Sirovica, Shen Ma, Shuang Ma, Simon Wang, Sulgi Kim, Suma Jayaram, Vaishaal Shankar, Varsha Paidi, Vivek Kumar, Xin Wang, Xin Zheng, Walker Cheng, Yael Shrager, Yang Ye, Yasu Tanaka, Yihao Guo, Yunsong Meng, Zhao Tang Luo, Zhi Ouyang, Alp Aygar, Alvin Wan, Andrew Walkingshaw, Andy Narayanan, Antonie Lin, Arsalan Farooq, Brent Ramerth, Colorado Reed, Chris Bartels, Chris Chaney, David Riazati, Eric Liang, Erin Feldman, Gabriel Hochstrasser, Guillaume Seguin, Irina Belousova, Joris Pelemans, Karen Yang, Keivan Alizadeh, Liangliang Cao, Mahyar Najibi, Marco Zuliani, Max Horton, Minsik Cho, Nikhil Bhendawade, Patrick Dong, Piotr Maj, Pulkit Agrawal, Qi Shan, Qichen Fu, Regan Poston, Sam Xu, Shuangning Liu, Sushma Rao, Tashweena Heeramun, Thomas Merth, Uday Rayala, Victor Cui, Vivek Rangarajan Sridhar, Wencong Zhang, Wenqi Zhang, Wentao Wu, Xingyu Zhou, Xinwen Liu, Yang Zhao, Yin Xia, Zhile Ren, and Zhongzheng Ren. 2024. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*.

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Albert Q. Jiang et al. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Norman P Jouppi et al. 2021. Ten lessons from three generations shaped google's tpuv4i. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE.

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

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

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

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2023. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*.

Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. 2020. Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training. *arXiv preprint arXiv:2001.04063*.

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

Apoorv Saxena. 2023. Prompt lookup decoding.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. *arXiv preprint arXiv:2207.07061*.

Benjamin Spector and Chris Re. 2023. Accelerating llm inference with staged speculative decoding. *arXiv preprint arXiv:2308.04623*.

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

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023a. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. 2023b. Spectr: Fast speculative decoding via optimal transport. *arXiv preprint arXiv:2310.15141*.

Yi Tay, Dara Bahri, Donald Metzler, et al. 2022. Scale efficiently: Insights from training and scaling large language models. *arXiv preprint arXiv:2210.03863*.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*.

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

Hugo Touvron et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. BloombergGPT: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

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.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183.

Hanling Yi, Feng Lin, Hongbin Li, Peiyang Ning, Xiaotian Yu, and Rong Xiao. 2024. Generation meets verification: Accelerating large language model inference with smart parallel auto-correct decoding. *arXiv preprint arXiv:2402.11809*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

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. *Preprint*, arXiv:2306.05685.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017a. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017b. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2023. Distillspec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv:2310.08461*.

A Application Specific Evaluation

On-device AI assistants require models that can efficiently generalize across downstream applications while operating under strict latency constraints. A common deployment paradigm involves fine-tuning base models for specific applications using parameter-efficient adaptation techniques such as LoRA (Hu et al., 2022; Tay et al., 2022). In contrast to conventional next-token prediction-based fine-tuning, we introduce a speculative fine-tuning objective for n-gram prediction, as detailed in Section 2.4, and fine-tune the shared adapters and stream embeddings for each downstream application (see Figure 2b).

Table 3 demonstrates the performance of Shared Speculative Streaming in comparison to existing methods. Our baselines include standard next-token prediction fine-tuning with LoRa adapters and inference with autoregressive decoding, as well as fine-tuned models paired with fine-tuned Medusa and Eagle speculative decoding heads on each application. Our evaluation metrics include wall-time speedup, mean accepted tokens, generation quality and additional parameter overhead. The results show that SS consistently achieves better acceptance rates and speedups while maintaining significantly lower parameter overhead compared to alternative approaches. Our method also demonstrates better wall-time latencies than traditional draft-target speculative decoding approach deployed on each downstream application as shown in Table 5. Notably, SS not only accelerates inference but also enhances generation quality compared to conventional next-token prediction fine-tuning. This positions our method as a compelling alternative to existing LoRA-based fine-tuning approaches employing a dedicated draft model or trained heads per application, offering both improved performance and reduced parameter overhead in speculative decoding settings. For detailed analysis of the draft-model based speculative decoding, we refer readers to Appendix N.

Although the parameter overhead of methods like Medusa may seem minimal relative to the base model size, it becomes increasingly significant as the number of downstream applications scales, especially in resource-constrained settings. We further discuss the importance of parameter efficiency in Appendix C.

B Learning Dynamics

In this section, we provide additional insights to complement the analysis in Section 3.2 and further elucidate why shared speculative streaming leads to improvements in both generation quality and decoding efficiency.

Gradient Flow Across Time. During training, speculative streams form a differentiable path through future residual approximations. This enables gradients from future tokens to influence earlier computations, which is not possible in standard autoregressive training. As the model learns to align speculative states with true future residuals, it benefits from temporally richer learning signals. This backward flow from speculative futures helps optimize the transformations in earlier layers to be more consistent with likely trajectories, improving token acceptance during inference. As shown in Figure 6, speculative supervision introduces gradient signals across early and mid-level layers suggesting that future-token losses influence earlier computations through the speculative paths.

Self-Distillation via Multi-Stream Attention. The multi-stream attention pattern in speculative streaming can be viewed as a form of internal self-distillation. Rather than relying solely on the current token's hidden state, the model integrates information across speculative futures through attention. This process resembles ensembling over short future rollouts, effectively consolidating predictive signals from multiple plausible paths. It leads to more stable and informed token decisions, especially under uncertain contexts. Figure 7 illustrates how multi-stream attention in speculative streaming resolves ambiguity in selecting a parallelization strategy by integrating contextual signals from near-future tokens.

Temporal Redundancy and Predictive Consistency. Natural language exhibits temporal redundancy: future tokens often confirm or disambiguate earlier ones. Speculative streaming leverages this redundancy by learning a residual transformation space where future tokens can be anticipated and approximated. This structure allows the model to reduce the mismatch between training and inference-time dynamics, improving the predictive consistency of the decoding trajectory.

Table 3: Comparison of wall-time speedup, mean accepted tokens, and parameter overhead across models of varying scales fine-tuned on downstream tasks. The mean accepted tokens metric serves as an accelerator-agnostic measure of speedup, representing the average number of accepted tokens per forward pass. Task-specific evaluation includes exact match accuracy for SqlCreateContext and ROUGE scores for DialogSum and E2E-NLG. The baseline corresponds to standard next-token prediction-based fine-tuning with LoRa adapters, whereas SS-Shared denotes speculative fine-tuning with shared adapters and embeddings, as detailed in Section 2.4 and Figure 2b. Notably, Medusa and Eagle heads are trained independently of the base model for each application, following (Cai et al., 2023; Li et al., 2024), resulting in identical downstream metrics to the baseline. In contrast, SS-shared jointly optimizes adapters for both next-token and future-token predictions, yielding improved downstream generation quality. Parameter overhead is reported relative to the baseline adapter overhead.

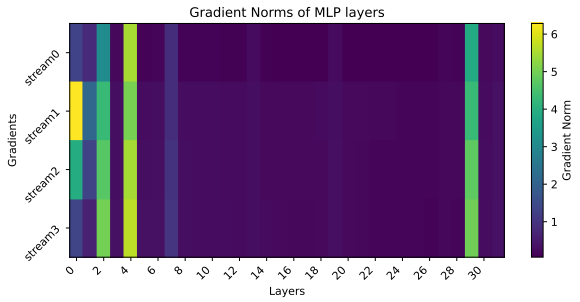
Dataset	Model	Method	SpeedUp (↑)	Mean Accepted Tokens (↑)	Metric (↑)	# Extra Parameters (↓)
SqlCreateContext	Mistral-Instruct-7B	Baseline	1.00	1.00	84.16	–
		Medusa	2.74	3.16	84.16	$5.9E8$
		Eagle	2.75	3.58	84.16	$2.4E8$
		SS-Shared	**2.93**	**3.67**	**84.50**	$8.2E4$
	PHI-3-Instruct-3.8B	Baseline	1.00	1.00	80.92	–
		Medusa	2.51	2.79	80.92	$4.3E8$
		Eagle	2.62	3.37	80.92	$1.3E8$
		SS-Shared	**2.92**	**3.65**	**84.10**	$6.1E4$
	Llama2-7b	Baseline	1.00	1.00	85.37	–
		Medusa	2.46	2.97	85.37	$5.9E8$
		Eagle	2.59	3.31	85.37	$2.4E8$
		SS-Shared	**2.81**	**3.57**	**85.93**	$8.2E4$
DialogSum	Mistral-Instruct-7B	Baseline	1.00	1.00	44.74/36.76	–
		Medusa	1.84	2.06	44.74/36.76	$5.9E8$
		Eagle	1.95	2.56	44.74/36.76	$2.4E8$
		SS-Shared	**2.04**	**2.96**	**44.89/37.09**	$8.2E4$
	PHI-3-Instruct-3.8B	Baseline	1.00	1.00	46.08/38.28	–
		Medusa	2.14	2.18	46.08/38.28	$4.3E8$
		Eagle	2.05	2.31	46.08/38.28	$1.3E8$
		SS-Shared	**2.32**	**2.85**	**46.30/38.32**	$6.1E4$
	Llama2-7b	Baseline	1.00	1.00	44.90/37.0	–
		Medusa	1.80	2.03	44.90/37.0	$5.9E8$
		Eagle	1.86	2.57	44.90/37.0	$2.4E8$
		SS-Shared	**1.90**	**3.05**	**45.0/37.85**	$8.2E4$
E2E-NLG	Mistral-Instruct-7B	Baseline	1.00	1.00	67.82/49.0	–
		Medusa	2.74	3.16	67.82/49.0	$5.9E8$
		Eagle	2.85	3.52	67.82/49.0	$2.4E8$
		SS-Shared	**2.93**	**3.67**	**68.37/49.09**	$8.2E4$
	PHI-3-Instruct-3.8B	Baseline	1.00	1.00	68.72/49.31	–
		Medusa	2.35	2.61	68.72/49.31	$4.3E8$
		Eagle	**2.42**	**2.76**	68.72/49.31	$1.3E8$
		SS-Shared	2.36	2.72	**69.38/50.22**	$6.1E4$
	Llama2-7b	Baseline	1.00	1.00	69.47/49.54	–
		Medusa	2.80	3.18	69.47/49.54	$5.9E8$
		Eagle	2.79	3.26	69.47/49.54	$2.4E8$
		SS-Shared	**2.89**	**3.38**	**69.52/49.93**	$8.2E4$

C Importance of Parameter Efficiency

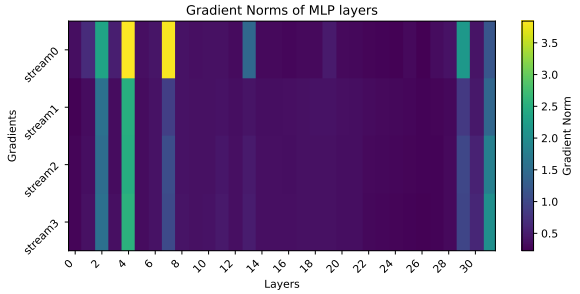
In production AI systems, such as Apple Intelligence (Gunter et al., 2024), BloombergGPT (Wu et al., 2023), ChatLaw (Cui et al., 2023), LoRA adapters are employed to fine-tune a shared base model across multiple downstream applications. In such systems, a dedicated draft model or Medusa heads are typically required for each downstream application to obtain high acceptance rates, resulting in substantial cumulative memory overhead. For instance, using Medusa heads with a Llama-2-7B model, which requires approximately 7.2 GB in 8-bit precision, introduces an overhead of 5.9×10^8 parameters per application, as shown in Table 3, amounting to 1.2 GB in 16-bit precision. When scaled to 10 applications, the cumulative overhead reaches 11.8 GB since each application requires independent heads. This surpasses the base model's footprint, making deployment extremely difficult in resource-constrained environments. Notably, the number of downstream applications can far exceed 10; for example, Apple Intelligence encompasses a suite of writing tools (Apple, n.d.), each necessitating dedicated adapters. In contrast, Speculative Streaming significantly reduces this overhead to just 8.2×10^4 parameters per application, enabling scalable and efficient multi-application deployment on consumer devices.

D Related Works

The inference speed of large language models (LLMs) is often constrained by the sequential na-



(a) Start of training (Step 0).



(b) Mid-training (Step 5000).

Figure 6: We visualize gradient norms across MLP layers due to speculative stream losses at the start of training and at step 5000. Early in training, speculative supervision injects strong gradients not only in later layers but also in early and mid-level MLP blocks, indicating that future-token predictions influence earlier computations via differentiable speculative paths. As training progresses, gradients become more structured and selective.

```
def main():
    directory = input("Enter the directory path: ")
    files = get_files_in_directory(directory)
    word_counts = Counter()
    with Pool() as pool:
        results = pool.map(process_file, files)
    for result in results:
        word_counts += result
    top_5_words = word_counts.most_common(5)
    print("Top 5 words with the most number of occurrences:")
    for word, count in top_5_words:
        print(f"{word}: {count}")
```

(a) Output from the auto-regressive (AR) model.

```
def main():
    directory = input("Enter the directory path: ")
    files = get_files_in_directory(directory)
    word_counts = Counter()
    with ThreadPoolExecutor() as executor:
        future_word_counts = {executor.submit(process_file, file): file for file in files}
        for future in concurrent.futures.as_completed(future_word_counts):
            word_counts += future.result()
    top_5_words = word_counts.most_common(5)
    print("Top 5 words with the most number of occurrences:")
    for word, count in top_5_words:
        print(f"{word}: {count}")
```

(b) Output from the speculative streaming (SS) model.

Figure 7: Speculative Streaming enables improved prediction through access to future context. Illustrated is a divergence point in decoding where both the AR and SS models initially generate identical code. At the point of selecting a parallelization strategy, the AR model predicts `Pool`, while the SS model, conditioned on upcoming context related to thread-based execution, predicts `ThreadPoolExecutor`. This example highlights how speculative streaming can guide the model towards more appropriate continuations by leveraging early glimpses of future context.

ture of auto-regressive decoding, which requires a complete forward pass of the network for each token generated. To mitigate the high inference latency, various strategies have been proposed to reduce the memory footprint of LLMs. Techniques such as model quantization (Frantar et al., 2022; Yao et al., 2022; Dettmers et al., 2023), knowledge distillation to smaller models (Gu et al., 2023; Agarwal et al., 2023b), and pruning (Frantar and Alistarh, 2023; Sun et al., 2023a) have emerged as effective solutions. More recently, Confident Adaptive Language Modeling (CALM) (Schuster et al., 2022) has introduced a method to dynamically adjust computational resources per token through early exiting in decoder layers. While CALM shows promise, it is hindered by issues related to key-value (KV) mismatch (Corro et al., 2023). To address the KV mismatch problem, skip decoding (Corro et al., 2023) allows for the bypassing of an increasing number of layers based on the position in the decoded sequence. While this approach eliminates KV mismatch, the predefined restrictions on the number of layers bypassed lead to suboptimal generation quality (Bhendawade et al., 2025). In contrast, speculative decoding methods provide a significant advantage over dynamic computing approaches, as they maintain generation quality while enhancing inference efficiency.

The original speculative decoding approach (Chen et al., 2023; Leviathan et al., 2023) utilizes a smaller draft model to generate a candidate sequence of tokens to be verified by the *target model*. Recent speculative decoding variants propose parallel computation along the batch axis (Sun et al., 2023b), and tree-structured batches (Miao et al., 2023; Spector and Re, 2023) to improve the acceptance rates of the guessed tokens by the target model and to further boost the performance. However, these methods encounter a common limitation: the necessity of developing an accurate and independent draft model for each downstream application. First, training such a draft model aligned with the main model is not trivial (Zhou et al., 2023). Second,

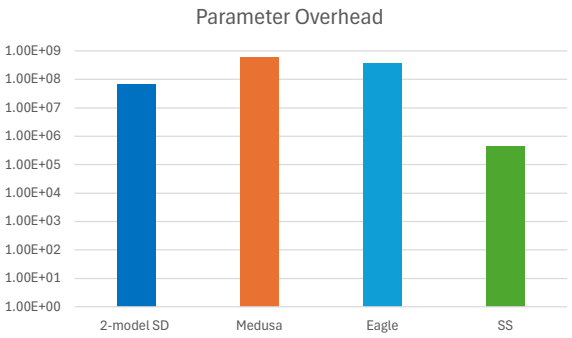


Figure 8: Parameter/Memory access overhead of different lossless speculative decoding architectures with Llama-13B.

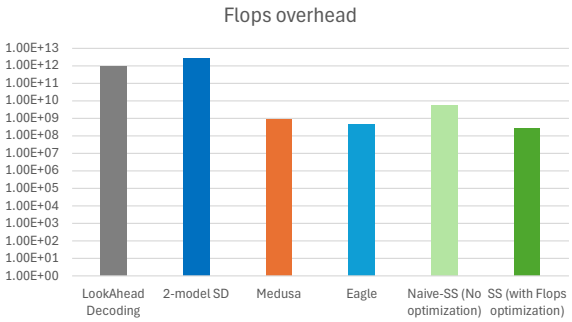


Figure 9: FLOP overhead of different lossless speculative decoding architectures per speculative draft generation with Llama-13B.

hosting two different models increases the system complexity, and is more computationally and operationally expensive to maintain as the number of applications increases.

Recently, single-model speculative decoding has gained attention. Inspired by (Qi et al., 2020; Stern et al., 2018), Medusa (Cai et al., 2023) extends the main model by training multiple output heads to predict future tokens in parallel. While Medusa eliminates the need for a separate draft model, each additional head introduces significant parameter overhead, making deployment challenging on resource-constrained devices. Furthermore, since speculated tokens are generated non-autoregressively, dependencies between them are not guaranteed, limiting practical speedups (Ankner et al., 2024). Hydra (Ankner et al., 2024) improves upon Medusa by incorporating an autoregressive draft head to enforce token dependencies. However, the small draft head size often leads to suboptimal speculation, and increasing its size results in similar autoregressive latency and parameter overhead issues similar to those observed in (Leviathan et al., 2023; Zhou et al., 2023).

Eagle (Li et al., 2024) refines these approaches by integrating a dedicated speculation layer within the target model. While this eliminates the need for an external draft model, its reliance on an autoregressive draft generation constrains speedup gains, and the additional speculation layer increases parameter overhead. Lookahead decoding (Fu et al., 2023) proposes a parallel decoding strategy without introducing new learnable parameters. While parameter efficiency is a key advantage, the non-learnable nature of the speculation process results in limited speedups. Prompt Lookup Decoding is another non-learnable strategy that circumvents additional model modifications by caching and retrieving token sequences from precomputed prompt-based lookups. While computationally lightweight, it struggles with generalization beyond cached sequences and suffers from increased retrieval latency when applied to long-tail distributions. SPACE (Yi et al., 2024) introduces structured speculative decoding by dynamically pruning infeasible predictions using syntactic constraints. However, its reliance on pre-specified constraints reduces flexibility across diverse decoding tasks. REST (He et al., 2024) employs reinforcement learning to optimize token speculation, but the additional training complexity and stability challenges hinder its practical adoption.

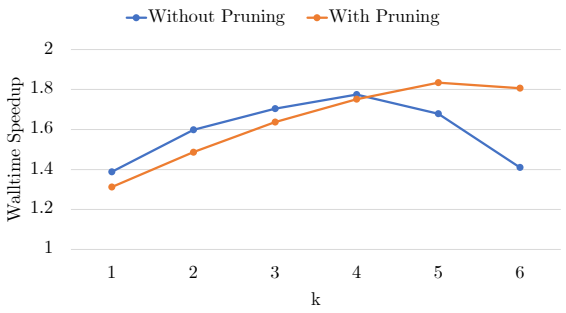


Figure 10: As more tokens (k) are sampled for tree drafting, speedup initially increases. This trend reverses as k continues to increase as the model transits to the compute-bound phase. Pruning less probable paths helps reduce compute, offering more speedup.

E Ablation:

E.1 Speculative Draft Size.

To improve the acceptance rate of the tree draft, we try various settings of γ, the number of speculative positions, and k, the number of sampled tokens per speculative position. Figure 10 shows wall-time

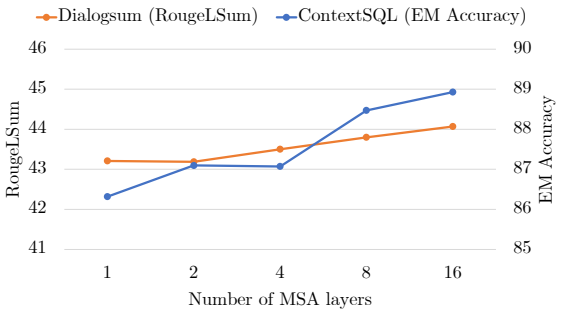


Figure 11: As the number of multi-stream attention layers increases, metrics on downstream tasks improves. Typically $N_s = 2$ to 8 yields a good trade-off between generation metrics and FLOPs overhead.

speedup for $\gamma = 3$. As we sample more tokens from each speculative position, advancement per forward pass, β increases since more candidates are available for verification, leading to more speedup. However, as we continue to increase k, forward pass latency overhead becomes more prevalent as the model transitions into compute-bound phase, ultimately reversing the speedup trend. This occurs because naively forming a tree draft leads to an exponential increase in batch size with k as described in 2.3. We insert a tree pruning layer to remove less probable paths and reduce the size of the tree draft. Pruning the tree draft reduces forward pass latency, and a well calibrated threshold ensures that only noisy paths in the tree get pruned. Tree pruning helps with wall-time speedup as k continues to increase as shown in Figure 10.

E.2 Number of MSA Layers

The number of MSA layers plays a crucial role in balancing generation quality and computational cost. Figure 11 presents the OPT-1.3B model's performance on Structured Query and Summarization tasks with increasing number of MSA layers. While more MSA layers improves performance, the additional FLOPs may outweigh the gains. Empirically, applying MSA to the top 2–8 layers achieves an optimal trade-off. We use 4 MSA layers for all experiments in Section 3.

E.3 Top-k Sampling

In the main paper, we reported speedup results using greedy sampling and $T = 0$. To further analyze speedups in the Top-k sampling regime, we evaluate various values of k and $T = 1$ for both Medusa and Speculative Streaming approaches. Figure 12 (b) shows the effect of increasing k on the walltime speedups and call reduction ratios[1]. Although increasing k leads to lower wall-time speedups for both baseline and target methods due to stochastic rejection of tokens, our approach retains its lead achieving better call reduction ratios and walltime speedups across different values of k.

E.4 Value Rotation

We analyzed more ways of differing computation of main stream from speculative streams. Apart from using dedicated stream embeddings, one way to differentiate the computation while incorporating a sense of relative position is simply rotating streams relative to each other. In this ablation, we initialize each stream with the main stream's hidden state and rotate the value projection during attention computation in the proportion of the relative distance from main stream as :

$$V_{tn}^k = V_t^k e^{i\epsilon n} \tag{8}$$

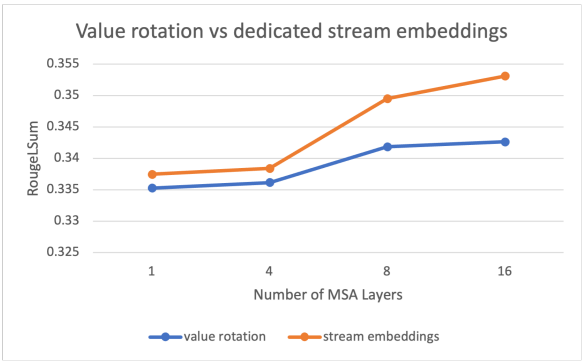
Where $1 <= n <= \gamma$ is stream index, V_t^k denotes value projection of main stream at time step t and layer k, while V_{tn}^k denotes value projection of stream n, $0 \leq \epsilon \leq \frac{\pi}{2N}$ denotes an arbitrary rotation step and N denotes the sum of maximum sequence length and number of streams. Figure 12 (a) shows the effect of using value rotation on ROUGE scores on the Dialog Summarization task with the OPT-1.3B model. Downstream metric for value rotation-based approach tends to be lower than using dedicated stream embeddings across different settings of MSA layers, however, the trend of increasing metric with added MSA layers remains the same. It is worth noting that for $N_s = 16$, simply rotating value projections achieves better metrics than using $N_s = 4$ with dedicated stream embeddings.

E.5 Effect of Quantization

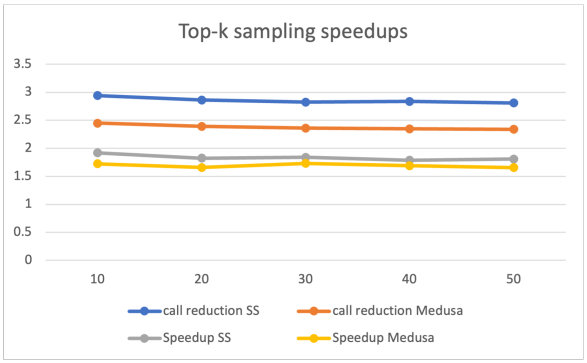
To investigate the effects of quantization, we perform experiments with 8-bit and 4-bit quantization using `bitsandbytes` (Dettmers et al., 2022) on the Text-to-SQL task (Zhong et al., 2017b) with Mistral-Instruct-7B. We compare the speedups achieved by speculative streaming (SS) to those of autoregressive decoding with baseline models employing the same quantization.

[1]The call reduction ratio represents the ratio of the number of 'model.forward()' calls required for autoregressive decoding to those required for speculative streaming. It is equivalent to the average number of tokens generated per 'model.forward()' call during target speculative streaming.

As demonstrated in Table 4, quantization alleviates memory bandwidth limitations inherent in autoregressive decoding, resulting in a reduction of speedup gains for speculative streaming compared to the autoregressive baseline. Despite this, SS still achieves a notable 2.7x speedup when using INT-4 quantization. Additionally, while some degradation in exact match metrics is observed with quantization, the speculative fine-tuning approach outlined in Section 2.4 continues to outperform next-token prediction-based fine-tuning in terms of downstream task performance.



(a)



(b)

Figure 12: (a) We analyze the effect of value projection rotation on ROUGELSum scores of the Dialog summarization task using OPT-1.3B as the base model for different numbers of MSA layers. Each stream is rotated in proportion to the distance from the main stream. (b) We study the effect of top-k sampling on wall-time speedups and call reduction ratios (mean tokens genearted per step) for Speculative Streaming (SS) and Medusa-style approaches using OPT-1.3B as a base model on the Meaning Representation task.

E.6 Breakdown of Speedup

Figure 13 presents an ablation of the primary mechanisms contributing to the efficiency of Speculative Streaming (SS). The base chain configuration leverages multi-stream attention between speculative and base streams, yielding substantial gains through linear speculative execution. Extending this to tree-mode amortizes target verification across multiple divergent hypotheses in the batch, enabling more aggressive parallelism and improving speedup. Finally, pruning introduces a low-cost early elimination of unlikely branches via early exiting, improving efficiency by reducing unnecessary compute.

Model	Quantization	Speedup	Metric
Baseline	FP16	-	84.16
	INT8	-	83.36
	INT4	-	81.08
SS-Shared	FP16	2.92	84.50
	INT8	2.84	83.89
	INT4	2.68	82.11

Table 4: Impact of quantization on the performance of Speculative Streaming

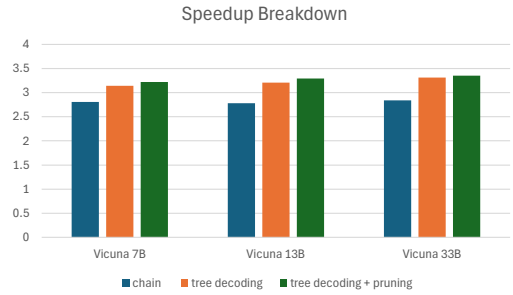


Figure 13: Ablation of effectiveness of Speculative Streaming components across Vicuna model scales. *Chain* denotes linear speculation via multi-stream attention as detailed in Section 2.1. *Tree decoding* enables parallel verification across batch trajectories as detailed in Section 2.2, while *Tree pruning* reduces redundant target verifications as detailed in Section 2.3. Together, these mechanisms compound to yield scalable speedup.

F Acceptance Criteria

We adopt the rejection sampling-based acceptance criterion proposed by (Chen et al., 2023) to mitigate distributional shift between the draft and target models. Specifically, we apply rejection sampling to select tokens from each path in the pruned tree (see Section 2.3), and the longest accepted path is used to advance decoding.To adhere to the principles of rejection sampling, we replace the draft model's output distribution by introducing a virtual distribution, which leverages speculative streams. More concretely, we replace the draft dis-

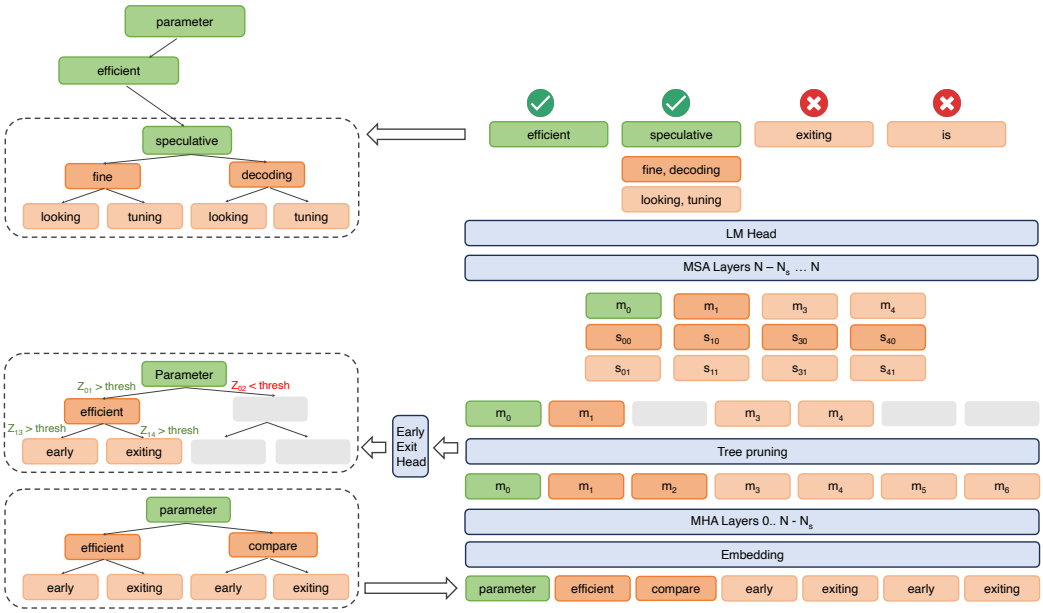


Figure 14: Parallel tree draft speculation and verification: Tree draft from the previous iteration is flattened for verification. After $N - N_s$ MHA layers, the tree pruning procedure obviates less probable tokens based on transition probability between parent and child tokens. In this illustration, "Z_i" denotes normalized early exit logits corresponding to main stream at index i, m_i, while "Z_{ij}" denotes transition probability between token at index i and j in flattened tree draft. The verification procedure is subsequently run on the pruned tree and speculative tokens are sampled from streams corresponding to the latest accepted token. In the above illustration, "speculative", "fine, decoding" and "looking, tuning" are sampled from streams m_1, s_{10} and s_{11}.

tribution $p(x \mid x_1, \ldots, x_{n+t-1})$ in Algorithm 2 of (Chen et al., 2023) with an augmented distribution $q(x \mid x_1, \ldots, x_n, s_{n0}, \ldots, s_{n(t-1)})$, where s represents the state from the prophet streams. Thus, our acceptance criterion is formulated as follows:

$$r < \min\left(1, \frac{q(x \mid x_1, \ldots, x_{n+t-1})}{q(x \mid x_1, \ldots, x_n, s_{n0}, \ldots, s_{n(t-1)})}\right), \tag{9}$$

where p and q represent the draft and target distributions from (Chen et al., 2023), $r \sim U[0, 1]$, and $1 \leq t \leq \gamma$.

G Implementation Details

G.1 Tree Draft Management

In this section, we go into more detail of tree draft sampling, flattening, and pruning. As shown in the main paper, when processing prompt $(x_1 ... x_t)$, we insert speculative streams along with the last token to generate logits, z_t corresponding to main stream and $(z_{t1} ... z_{t\gamma})$ corresponding to speculative streams. Tree draft is sampled following the procedure described in Section 2.2. The sampled draft is then flattened along the sequence length dimension and the attention mask is composed such that child nodes attend to their predecessors starting with root as shown in Figure 14 and Figure 15. The root token of the tree draft is the correction issued by main stream. Each iteration after prompt processing involves verifying the previous tree draft and sampling a new one. After passing the tree draft through $N - N_s$ layers, we use contextual features learned by middle layers to approximate transition probability between parent and child tokens. As shown in Figure 14, since the transition probability between token "parameter" and "compare" is less than a set threshold, we prune the sub-tree starting from "compare" in the feature domain , and m_2, m_5, m_6 are pruned. Notably, the key value cache of layers before the pruning layer is not trimmed at this point to keep pruning latency overhead minimal. Key value cache backtracking is done lazily after each generation step.

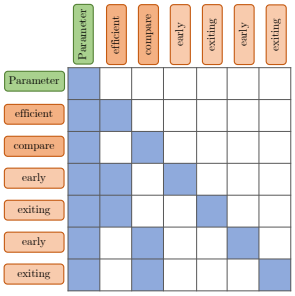


Figure 15: The attention mask for the tree draft is composed in such a way that child tokens can attend to all predecessors starting from the root, with the root being the correction issued by the main stream. In this illustration, "early" attends to "parameter" and "efficient" and itself, as "parameter - efficient - early" forms one path in the tree. "early" is also replicated to form another path, "parameter - compare - early". This attention mask allows batching multiple paths and increases the acceptance rate as the number of candidates increases.

Speculative streams are inserted alongside each node in the pruned draft. Layers $(N - N_s..N)$ use multi-stream attention as described in Equation (3) and Equation (2). The verification procedure finds the longest matching path in the pruned tree that main stream can accept. As shown in Figure 14, path ("parameter", "efficient", "speculative") is accepted. Correction token sampled from logits of main stream corresponding to last accepted token, m_1 becomes new root while tokens sampled from logits of streams (s_{10}, s_{11}) form the sub-tree.

H FLOPs Optimization

Naively implemented, Speculative Streaming incurs higher FLOP overhead compared to Eagle. It is worth noting that modern accelerators demonstrate compute bandwidth that exceeds memory access bandwidth by an order of magnitude or more (Agarwal et al., 2023a; Jouppi et al., 2021), meaning increased FLOPs do not necessarily translate to increased decoding latency. Nevertheless, to ensure fair comparison and efficiency in compute-bound scenarios, we introduce targeted optimizations.

H.1 Attention FLOPs Optimization

For medium-to-long context lengths, attention computation dominates FLOPs in the self-attention layer, surpassing the contribution from QKV projection layers. Specifically, matrix multiplications involving queries, cached keys, and cached values scale with $l_{kv} \times l_q$ where l_{kv} denotes previous context length and l_q denotes current query length. Since Speculative Streaming pairs speculative streams with base streams, a naive implementation results in more FLOPs compared to a standard attention layer. To address this, we limit the attention of speculative residual streams to selectively attend to the top p most relevant tokens identified by the base residual stream based on top attention coefficients[2]. This is possible since base and speculative residual streams are processed in same forward pass and speculative streams have access to attention coefficients of base stream. Note that, each of the speculative streams still retains the flexibility to assign distinct attention coefficients to these tokens, optimizing residual transformation at corresponding positions.

H.2 MLP FLOPs Optimization

The stream adapters operating on the speculative residual stream are intentionally designed with lower rank to reduce FLOP overhead by a factor proportional to h/R_s, where h denotes hidden size of base stream and R_s denotes rank of stream adapter. We set $R_s = 8$ to achieve good acceptance rates while keeping FLOPs and parameter overhead minimal.

These optimizations significantly reduce the FLOP overhead per speculative draft generation, as illustrated in Figure 9. We include these optimizations for all experiments involving Speculative Streaming, as detailed in Section 3.

I Segment Attention

Naive training with Speculative Streaming increases the batch dimension along the sequence length axis by a factor of γ, resulting in attention computation reaching peak memory usage with larger batches. To address this issue, we propose a segment-based attention method that significantly reduces peak memory consumption while enhancing training throughput. We divide each training sample into a prompt and multiple segments of completion. Since each stream corresponding to each token must attend to the previous streams of the same token as well as to the prompt and previous completion tokens, we can eliminate the need for prompt streams in our design. Furthermore,

[2]We set to $p = 64$ and attend to top 64 tokens as identified by the base residual stream.

Table 5: Mean walltime latency per sample and generation metrics comparison with standard draft-target (Two-model) speculative decoding approach using OPT-125m as the draft model. Draft model is fine-tuned on each application.

Dataset	Target	Method	Target calls	Draft Calls	Walltime Latency ($ms \downarrow$)	Metric ($\uparrow$)
SqlContext	OPT-1.3b	Two-model SD	6.59	22.35	269.24	84.98
		SS-Shared	7.79	0	**133.48**	**87.40**
	OPT-6.7b	Two-model SD	6.60	22.41	301.10	89.13
		SS-Shared	6.88	0	**157.04**	**89.34**
Dialogsum	OPT-1.3b	Two-model SD	11.65	42.59	493.59	43.40/35.60
		SS-Shared	13.41	0	**248.26**	**44.07/35.99**
	OPT-6.7b	Two-model SD	12.15	35.76	555.99	44.40/36.60
		SS-Shared	14.45	0	**444.67**	**44.42/36.81**
E2E-NLG	OPT-1.3b	Two-model SD	8.86	31.47	345.72	**69.48**/50.17
		SS-Shared	9.80	0	**164.23**	69.32/**50.51**
	OPT-6.7b	Two-model SD	8.90	31.58	412.02	69.34/**49.88**
		SS-Shared	10.31	0	**244.80**	**69.45**/49.78

by segmenting the completion, we retain only the streams associated with the required segments in memory, as illustrated in Figure 16. This design significantly reduces peak memory consumption and ensures the scalability of our approach when training with larger batch sizes, ultimately yielding improved throughput.

J Training cost

Since Speculative Streaming is parameter-efficient, training involves fine-tuning only the LoRA parameters of stream adapters and embeddings. The training time is comparable to that of training Medusa heads. We fine-tuned the Vicuna-7B model on the ShareGPT dataset in approximately 4 hours using segment attention, which is comparable to the 3–4 hours required for training Medusa heads. Additionally, we successfully trained Vicuna-33B models on a single 80-GB GPU for one epoch by loading the base model in NF4 precision and keeping only the adapters of 4 MSA layers in full precision, completing training in approximately 7 hours. Fine-tuning for application-specific tasks (see Table 3) is relatively faster, requiring approximately 1-3 hours per application based on dataset size.

K Compute and Memory Profiling

The draft overhead associated with the standard draft-target speculative decoding approach tends to be non-trivial especially when the latency ratio between target and draft models $c_{target}/c_{draft} <= 10$. This is because speculation and verification procedures are run in a serial manner. Figure 17 shows the kernel utilization timeline when OPT-125m is used as a draft while OPT-1.3b model is used as the target. Auto-regressive draft generation decreases overall kernel utilization in draft-target approach, while additional computation involved in MSA layers increases kernel utilization in case of Speculative Streaming (see Figure 19) thereby efficiently utilizing the accelerator and speeding up the decoding process. Negligible-cost draft models may offer a better choice to keep kernel utilization at higher levels in case of draft-target approach, however, acceptance rates tend to drop as draft model size decreases.

L Batching

All the results presented in Section 3 are with batch size of 1 for on-device setup. We also experiment with batching for server setup where queries from multiple users are batched to increase throughput and accelerator utilization. To achieve maximum throughput with batching, we disable tree decoding and tree pruning and use only best speculated path for each decoding step for every sequence in a batch. Since our method primarily relies on utilizing flops to accelerate decoding, with batching we do see some degradation in speedup per sample as depicted in Figure 18, however we consistently achieve >2X speedups while keeping throughput same as batched autoregressive decoding.

M Recommended Hyperparameters

Our experiments indicate that the following setup yields robust performance, achieving substantial speedups across different tasks and model sizes:

- Number of streams: 4
- Number of MSA layers: 4

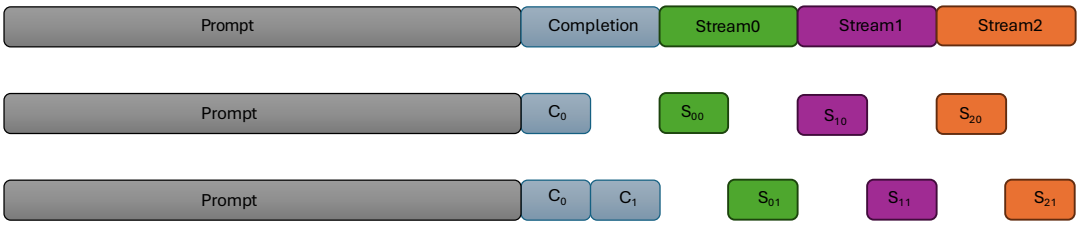
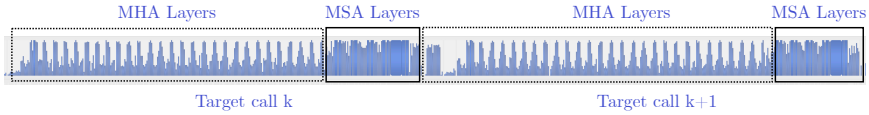
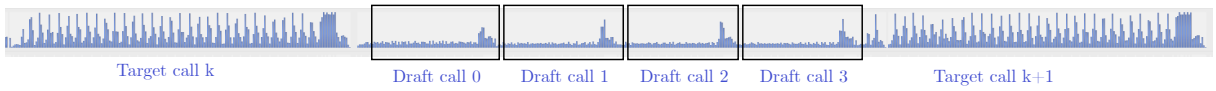


Figure 16: Streams corresponding to prompt are not required while training. Completion is divided into multiple segments and streams of each segment only attend to previous streams from same segment and main stream of previous segments. Uncolored portion indicates those tokens/streams are not required to be kept in memory.



(a) Speculative Streaming



(b) Two Stage Speculative Decoding

Figure 17: Kernel utilization timeline for speculative streaming and the standard draft-target speculative decoding. Draft-target approach runs speculation and verification in serial manner while it is parallelized in Speculative Streaming. Auto-regressive draft generation often has low GPU utilization leading to decreased overall kernel utilization while MSA layers in Speculative Streaming increase kernel utilization by generating a non-autoregressive draft and speeding up decoding significantly.

- Stream adapter rank: 8
- Tree Factor: 3
- $\alpha_0 = 1, \alpha_1 = 0.1$ for shared mode
- $\alpha_0 = 0, \alpha_1 = 1$ for lossless mode

N Analysis of 2-model speculative decoding

Speculative Streaming consistently achieves significantly lower walltime latency than standard draft-target speculative decoding as depicted in Table 5. It is worth noting that, target model calls of draft-target speculative decoding are slightly lower than Speculative Streaming, however, it comes at the cost of auto-regressively running draft model γ times to generate speculative draft. On the other hand, draft generation with Speculative Streaming incurs almost no additional latency overhead, as target model decoding tends to be memory-bound even with increased tree draft size. This translates to increased kernel utilization and arithmetic intensity as shown in Figure 19.

An argument could be made that a smaller draft model may perform better since drafting should cost less, but acceptance rates may drop as well as the draft model size is decreased. To formalize the comparison with standard draft-target speculative decoding, we do the following analysis, suppose, C_{draft} is the latency cost associated with forward pass through the draft model, C_{target} is the cost associated with forward pass through target model, while C_{ss} is cost associated with speculative streaming forward pass. ζ is the number of decoding tokens advanced during the verification step for the draft-target approach while β is the number of tokens advanced in Speculative Streaming. We equate latency cost associated with single token advancement to compare both approaches.

$$(\gamma * C_{draft} + C_{target})/\zeta = C_{ss}/\beta \quad (10)$$
$$(\gamma + C_{target}/C_{draft})/\zeta = (C_{ss}/C_{draft})/\beta$$

Assuming $\gamma = 4, C_{target}/C_{draft} = 10$, and $C_{ss} \approx C_{target}$, $\zeta = 1.4\beta$, meaning that advancements per verification step in standard draft-target approach have to be 1.4X of Speculative Streaming to achieve wall time latency parity. Note that, this analysis ignores cache adjustment overhead and prompt processing overhead, but provides valuable intuition to guide the choice between draft-target vs Speculative Streaming approaches. We also analyze under which settings speculative streaming is likely to offer more benefits as compared to the

Table 6: Comparison of mean accepted tokens of various speculative decoding approaches across Llama and Vicuna models of varying scales on MT-Bench. Methods such as 2-Model SD, Eagle generates speculative draft in auto-regressive (AR) manner while Medusa and Speculative Streaming generate speculative draft in a non auto-regressive (NAR) manner. Hydra drafts are generated in a semi auto-regressive (SAR) manner. Results for Medusa, Medusa-2, Hydra, Eagle, and LookAhead decoding are taken from their respective papers, with Hydra results corresponding to the best-performing variant, Hydra++.

Model	Method	Speculation Generation	Mean Accepted Tokens (↑)
Vicuna-7B	2-Model SD	AR	3.46
	Medusa	NAR	2.67
	Hydra	SAR	3.70
	Medusa-2	NAR	3.47
	Eagle	AR	3.94
	SS-Lossless	NAR	3.52
	SS-Shared	NAR	3.68
Vicuna-13B	2-Model SD	AR	3.67
	Medusa	NAR	2.72
	Hydra	SAR	3.72
	Medusa-2	NAR	3.51
	Eagle	AR	3.98
	SS-Lossless	NAR	3.63
	SS-Shared	NAR	3.76
Vicuna-33B	2-Model SD	AR	3.54
	Medusa	NAR	2.56
	Hydra	SAR	3.62
	Medusa-2	NAR	3.01
	Eagle	AR	3.68
	SS-Lossless	NAR	3.75
	SS-Shared	NAR	3.78
Llama-2-Chat-7B	2-Model SD	AR	3.58
	LookAhead	NAR	2.08
	Eagle	AR	3.62
	SS-Lossless	NAR	3.46
	SS-Shared	NAR	3.58
Llama-2-Chat-13B	2-Model SD	AR	3.55
	LookAhead	NAR	1.87
	Eagle	AR	3.90
	SS-Lossless	NAR	3.77
	SS-Shared	NAR	3.86

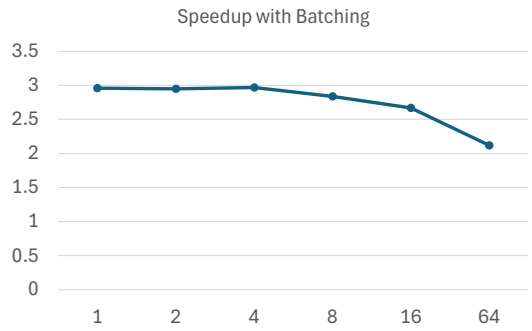


Figure 18: Walltime speedup for different batch sizes on MT-Bench with Vicuna-7B Model.

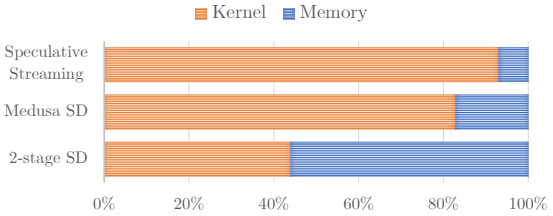


Figure 19: Kernel and Memory utilization comparison on Nvidia A100.

standard draft-target approach. Fig. 20 shows theoretical speedups of Speculative Streaming over draft-target based approach for different Target to draft latency ratios. As the latency ratio increases, the draft-target approach is likely to offer more speedup benefits when $\zeta/\beta > 1$, meaning that when the draft model is accurate enough to achieve more token advancements per target model verification step than Speculative Streaming and also small enough to yield higher latency ratios, it is likely to benefit more. Creating such a model usually requires significant engineering effort. In downstream application settings, finding ideal draft models becomes even more challenging since ζ tends to vary based on application. If applications share the draft model and only train adapters, the draft model may not remain small enough to meet target-to-draft latency ratios, making it challenging to achieve better speedups.

N.1 Experimental Setup Details

For experiments described in Section 3, our recipe involves training stream adapters and embeddings in BFloat16, using the AdamQ optimizer, a learning rate of 5e-4, and a linear scheduler. For tree pruning (see Section 2.3), we use a low-rank linear transformation of rank 8 to keep parameter overhead minimal. We experimented with linear transformations of different ranks to initialize speculative streams from main stream as described in Equation (4), however we find that simply using identity transformation achieves similar performance with much less parameter overhead. We use identity transformation for all the experiments described in Section 3. We compare MT bench and SpecBench speedups of our approach with best baseline configurations from corresponding papers. In Application specific settings (see Table 3), we report best results for Medusa and our approach over different γ and k values. We pass 32 nodes as a tree draft for speculative streaming after the pruning layer while in case of Medusa we pass 64 nodes, as these configurations yield the best walltime speedups for respective approaches. We use 'hard' matching criteria for verification of speculative draft. Relaxing this criterion to 'soft' matching may yield higher speedups (Cai et al., 2023) but may compromise generation quality. We defer this exploration to future work. In application spe-

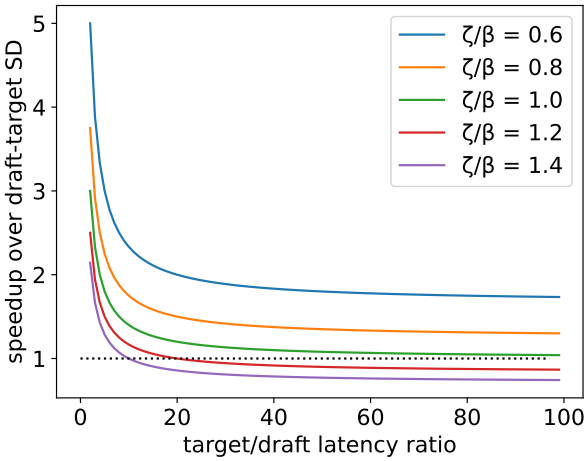


Figure 20: Speedup: Speculative Streaming speedups over draft model based speculative decoding for different ζ/β and target/draft latency ratios, where ζ denotes the number of advancements per verification step for draft model based speculative decoding while β denotes the same for Speculative Streaming.

cific settings, both Medusa heads and the number of maximum streams are fixed to 4 and the residual blocks per head used in Medusa are set to 1. Since Eagle requires autoregressive drafting, we report the speedups in Table 3 considering optimal autoregressive steps that balance speculation and verification latencies and achieve best speedup. For comparison with standard draft-target speculative decoding (Leviathan et al., 2023), we use OPT models since they come with different configurations and sizes. OPT-125m is deployed as a draft model while OPT-1.3b and OPT-6.7b are used as target models since a ratio of 10-100X is typically considered to be optimal. We compare our approach with LookAhead decoding using best configuration reported in (Fu et al., 2023).

O Long Context Experiments

To evaluate performance on long sequences, we trained lossless speculative streaming on the Arxiv-summarization dataset (Cohan et al., 2018) and tested it on the Summarization task from the LongBench dataset (Bai et al., 2023). Since the KV cache is shared between the main and speculative streams, there is no additional runtime memory overhead associated with longer contexts. While compute in attention layers increases due to longer context, the compute in MLP layers remains the same, and decoding is still memory bandwidth bound. We achieved a 2.64X speedup on the Summarization test set using gamma = 3 and k = 4.

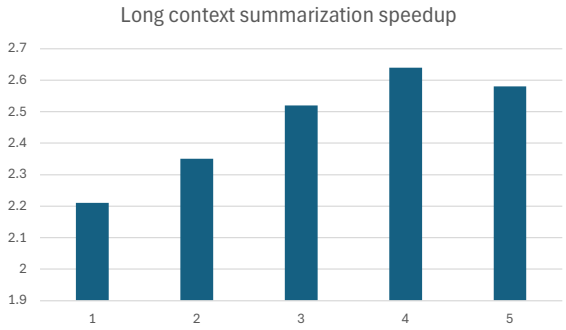


Figure 21: Speedups for long-context summarization tasks with varying top-k tokens sampled during drafting.

P Qualitative Examples

In this section, we present qualitative examples to illustrate the effectiveness of Speculative Streaming. By examining specific instances, we aim to highlight how this approach enhances the overall performance of the decoding process. An example of the SQL query generation task is shown in Figure 22, while a dialog summarization example is shown in Figure 23. Each row indicates the previous sequence of accepted draft tokens (in black) and the new sequence of generated tokens in green/red. We use $\gamma = 4$ and $k = 1$ to illustrate the decoding process. Green tokens in each row indicate tokens accepted in the forward pass, while red tokens indicate tokens rejected in the forward pass. Speculative Streaming generates meaningful drafts with high acceptance rates by capturing dependencies between tokens quite effectively, despite generating them in a non-auto-regressive manner.

SELECT in _ count y

SELECT in _ count y _ tu ition _ per

SELECT in _ count y _ tu ition _ per_ credit _ ~~credit~~ _

SELECT in _ count y _ tu ition _ per_ credit _ hour __ fall _ _

SELECT in _ count y _ tu ition _ per_ credit _ hour __ fall _ 2009 _ FROM table _

SELECT in _ count y _ tu ition _ per_ credit _ hour __ fall _ 2009 _ FROM table _ 22 30 88 81 _

SELECT in _ count y _ tu ition _ per_ credit _ hour __ fall _ 2009 _ FROM table _ 22 30 88 81 _ 2 WHERE college = "

SELECT in _ count y _ tu ition _ per_ credit _ hour __ fall _ 2009 _ FROM table _ 22 30 88 81 _ 2 WHERE college = " Mer ~~Er " College <\s>~~

SELECT in _ count y _ tu ition _ per_ credit _ hour __ fall _ 2009 _ FROM table _ 22 30 88 81 _ 2 WHERE college = " Mer Cer " <\s>

Figure 22: Speculative streaming on SQL generation task for $\gamma = 4$ and $k = 1$, each pass verifies the previous draft and generates a maximum of 5 tokens. For instance in pass 4, "$credit$" and "_" (shown in red) are rejected and "$hour$", "_", "$fall$", "_", "_" are speculated.

Person 2 # ~~and~~

Person 2 # thinks Lincoln ~~is a character~~

Person 2 # thinks Lincoln was a ~~character and he~~

Person 2 # thinks Lincoln was a man of ~~character and he~~

Person 2 # thinks Lincoln was a man of sound character and # person

Person 2 # thinks Lincoln was a man of sound character and # person 1 # adm ires him

Person 2 # thinks Lincoln was a man of sound character and # person 1 # adm ires him for his courage and ~~and~~

Person 2 # thinks Lincoln was a man of sound character and # person 1 # adm ires him for his courage and rights and humility . </s>

Figure 23: Speculative streaming on Dialog Summarization task for $\gamma = 4$ and $k = 1$, each pass verifies the previous draft and generates a maximum of 5 tokens. For instance, in pass 3, "is", "a", "$character$" are rejected and "was", "a", "$character$", "and", "he" are speculated.