STree: Speculative Tree Decoding for Hybrid State-Space Models

Yangchao Wu[1*] **Zongyue Qin**[1] **Alex Wong**[2] **Stefano Soatto**[1]

[1]**UCLA** [2]**Yale University**

Abstract

Speculative decoding is a technique to leverage hardware concurrency in order to enable multiple steps of token generation in a single forward pass, thus improving the efficiency of large-scale autoregressive (AR) Transformer models. State-space models (SSMs) are already more efficient than AR Transformers, since their state summarizes all past data with no need to cache or re-process tokens in the sliding window context. However, their state can also comprise thousands of tokens; so, speculative decoding has recently been extended to SSMs. Existing approaches, however, do not leverage the tree-based verification methods, since current SSMs lack the means to compute a token tree efficiently. We propose the first scalable algorithm to perform tree-based speculative decoding in state-space models (SSMs) and hybrid architectures of SSMs and Transformer layers. We exploit the structure of accumulated state transition matrices to facilitate tree-based speculative decoding with minimal overhead relative to current SSM implementations. Along with the algorithm, we describe a hardware-aware implementation that improves naive application of AR Transformer tree-based speculative decoding methods to SSMs. Furthermore, we outperform vanilla speculative decoding with SSMs even with a baseline drafting model and tree structure on three different benchmarks, opening up opportunities for further speed up with SSM and hybrid model inference. Code can be find at: `https://github.com/wyc1997/stree`.

1 Introduction

Recursive sequence models, such as autoregressive (AR) Transformers, produce a single token with each forward pass. Speculative decoding is a technique to make this process more efficient by leveraging a smaller 'draft model' to generate multiple tokens, and a separate 'verifier model' to validate the proposed drafts [13]. The efficiency stems from exploiting the concurrency of parallel computer hardware to verify multiple tokens in a single model call, while ensuring that the accepted drafts are identical to those that would have been generated by repeated calls to the original model. AR Transformers leverage a sliding window of input tokens (context) as their 'state', which is ideal for speculative decoding since their context can be easily edited to remove unverified tokens.

On the other hand, State-Space Models (SSMs) maintain an explicit Markov state that is designed and trained to be a sufficient statistic of the past for the purpose of predicting one-step ahead [17]. They are not naturally suited for speculative decoding, since the past states are discarded and the verifier would need to backtrack each speculated state, hampering the efficiency of the whole process. Recently, hardware-aware efficient methods for speculative decoding in SSMs have been proposed [24, 25], but they do not leverage tree-based verification that drives the most efficient methods for

*Correspondence to `wuyangchao1997@g.ucla.edu`

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

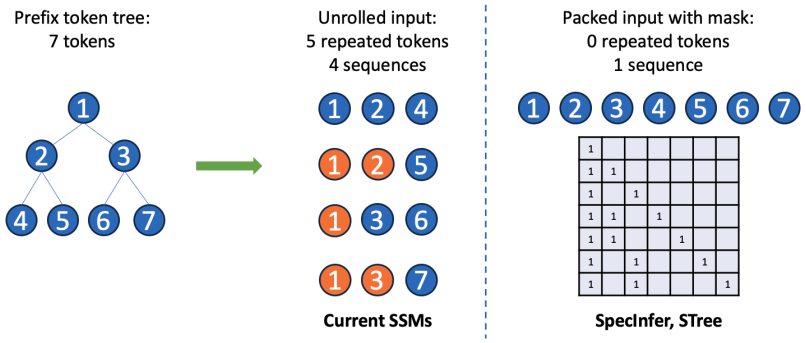


Figure 1: Methods to decode a prefix token tree with AR Transformers. A prefix token tree can be unrolled into multiple sequences (used by current SSMs) and computed as a batch or packed into one sequence using a mask to indicate the tree structure (first used by SpecInfer [19] and extended to SSMs here). The former leads to inefficiency due to repeatedly computed tokens (in orange).

AR Transformers. This paper proposes the first algorithm to do so, with a method that is applicable to both SSMs and hybrid architectures interleaving SSMs and Transformer layers.

To perform tree-based verification in AR Transformers [19, 14, 4, 2, 23, 21, 22], a prefix token tree is built with the draft model and verified with the target model. Existing works pack the token tree into one sequence and leverage a topology-aware mask [19] with self-attention to compute the output of the tree in one model call (Fig. 1 Right), thus avoiding repeated computation when naively unrolling the tree into individual sequences.

However, modern SSM realizations [6, 9] are designed to scan through all tokens in the past causally to obtain the state, lacking a mechanism to specify the tree structure. As a consequence, current SSMs can only use the unrolled input (Fig. 1 Left), leading to inefficient repeated token generation. Moreover, SSMs require one state per input sequence in the batch, leading to an explosion of the memory footprint when using the unrolled input. As the size of the tree grows, unrolling the tree quickly becomes infeasible. Therefore, we seek ways to leverage the characteristics of the hardware to perform multiple steps of state update following the tree structure through a single pass through the model.

We propose State-Space Speculative Tree (STree) decoding, a method to facilitate tree-based speculative decoding in SSMs through accumulating state transition matrices according to the tree structure. The tree is easily computed with minimal overhead relative to current SSM state update implementations. We describe a hardware-aware implementation that already in its simplest instantiation shows improvement over the baseline naive application of tree-based speculative decoding to SSMs.

Our contributions in this paper are to (i) propose what, to the best of our knowledge, is the first scalable method to leverage tree decoding in the speculative decoding for both SSMs and hybrid architectures; we also (ii) provide a simplified analysis of the trade-off between acceptance length and model runtime to help determine whether we should scale tree size or even use tree decoding. Finally, we (iii) empirically demonstrate that with a baseline drafting model and static tree structure, there are already improvements in generation speed, thus opening the door to further investigation of more advanced speculative decoding methods employed with transformers.

2 Related Work

State-space models (SSMs) are sequence models that maintain a hidden state and use it to predict the next datum in the sequence given the previous ones [17]. Such models can be stacked, so the output of one is the input of another [10, 26], and their parameters can be input-dependent [12, 9, 27]. In order to scale model size, transition matrices are often restricted to be diagonal [6, 7], while to leverage efficient hardware-aware implementations, SSM layers are often interleaved with Transformers in Hybrid architectures [16, 8], also useful for initial distillation [24]. Our method is

based on scalable SSMs of the general form (1).

$$\begin{cases} x_{t+1} = A(u_t)x_t + B(u_t)u_t \\ y_t = C(u_t)x_t + D(u_t)u_t \end{cases} \tag{1}$$

Speculative decoding denotes a set of techniques designed to parallelize sequential inference in large language models (LLMs) by utilizing a smaller 'draft model' to produce multiple candidate trajectories and a large 'verifier model' to test them for consistency with the original model [13]. Sub-networks can also be used for drafting, a form of 'self-speculation' [18], while speculation can be applied to latent features [15, 14] in addition to the output. Structural changes to the speculative processes include tree-based verification [20, 4], multi-head decoding [2], and beam sampling[21, 22]. The same methods can also be used for 'lossy verification' using judge decoding [1].

Tree-based verification [19] has been shown effective in improving the acceptance length of the drafted sequence [15, 14, 4, 2, 23, 21, 22]. EAGLE [15] reported an increase in acceptance length by using a static tree structure, leading to an overall speed up ratio improvement. Sequoia [4] generates the optimal tree for the speculated tokens using a dynamic programming algorithm and achieved further improvement. The underlying mechanism that enables tree-based verification is Transformers' ability to specify which tokens to attend to with an arbitrary attention mask individually for each token. A prefix token tree can be packed into a sequence and a topology aware mask [19] can be used to inform the self attention block the tree structure.

Speculative SSMs have been independently championed by [24, 25]. Since the state in SSMs is updated causally to summarize the past history, extending speculative decoding to SSMs require backtracking the state, which is non-trivial at scale. Both [24, 25] proposed hardware-aware efficient algorithms to backtrack the state while performing the forward pass. However, neither leveraged tree-based verification, since currently available algorithms for SSM updates require unrolling the tree into individual sequences, which leads to inefficiencies. These include repeated token computation and extra states to be maintained.

3 Method

Formalization. Given a prefix token tree T with tokens $\{t_1, \ldots, t_N\}$ as vertices and t_1 as the root node, we can pack the tokens into one sequence $S = \{t_1, \ldots, t_N\}$ and represent the topology of the tree using a special attention mask $L \in \{0, 1\}^{N \times N}$ ('tree' mask). Each vertex in T has one unique path to the root node, and we denote these paths as $s_i = \{t_n | t_n \text{ is in the path from } t_1 \text{ to } t_i\}$. Then the tree mask L can be constructed by the indicator function:

$$L_{i,j} = \mathbb{1}_{s_i}\{t_j\}. \tag{2}$$

Given the pre-SSM input features $u = \{u_1, \ldots, u_N | u_i \in \mathbb{R}^{d_u}\}$ of the packed sequence S and an initial state $x_0 \in \mathbb{R}^{d_x}$, our goal is to compute the output sequence $y = \{y_1, \ldots y_N | y_i \in \mathbb{R}^{d_u}\}$ without repeatedly computing any tokens or requiring extra SSM states. We present our approach below.

Tree decoding with State-space models. The input feature u is first mapped onto the parameters using a linear projection. Here we abuse the notation to let $X_t := X(u_t)$:

$$A_t = W_A u_t \in \mathbb{R}^{d_x \times d_x} \quad B_t = W_B u_t \in \mathbb{R}^{d_x \times d_u} \quad C_t = W_c u_t \in \mathbb{R}^{d_u \times d_x} \tag{3}$$

Ignoring the last term D_t in Eqn.(1) for simplicity of notation, for each $y_t = C_t x_t$, we can expand it recursively into:

$$\begin{aligned} y_t &= C_t(A_t^{L_{t,t}} A_{t-1}^{L_{t,t-1}} \ldots A_1^{L_{t,1}} x_0 + L_{t,1} A_t^{L_{t,t}} A_{t-1}^{L_{t,t-1}} \ldots A_2^{L_{t,2}} B_1 u_1 + \cdots + B_t u_t) \\ &= C_t(A_t^{L_{t,t}} A_{t-1}^{L_{t,t-1}} \ldots A_1^{L_{t,1}} x_0 + \sum_{s=1}^{t} L_{t,s} (\prod_{j=s+1}^{t} A_j^{L_{t,j}}) B_s u_s) \end{aligned} \tag{4}$$

As done in [9, 6, 7, 16], we enforce a diagonal structure on A_i to reduce the products in Eqn.(4) to a sum of logarithms:

$$\begin{aligned} y_t &= C_t(\exp\{\sum_{i=1}^{t} L_{t,i}\log(A_i)\}x_0 + \sum_{s=1}^{t} L_{t,s}\exp\{\sum_{j=s+1}^{t} L_{t,j}\log(A_j)\}B_s u_s) \\ &= C_t\exp\{\sum_{i=1}^{t} L_{t,i}\log(A_i)\}x_0 + \sum_{s=1}^{t} L_{t,s}\exp\{\sum_{j=s+1}^{t} L_{t,j}\log(A_j)\}C_t B_s u_s \end{aligned} \tag{5}$$

Since A_i is diagonal, we can represent the diagonal of $\log A_i$ as a vector and assemble a new matrix $A_{log} = [diag(\log A_1), \ldots, diag(\log A_N)]^T \in \mathbb{R}^{N\times d_x}$. We define:

$$A_{tree} := (LA_{log}) \quad (A_{tree})_t = \sum_{i=1}^{N} L_{t,i} \times diag(\log A_i) \in \mathbb{R}^{d_x} \tag{6}$$

We note that $(A_{tree})_t$ is equivalent to $\sum_{i=1}^{t} L_{t,i}\log(A_i)$ in Eqn.(5), since A_i is diagonal and we can always organize the tokens in the sequence such that $L_{t,i} = 0 \quad \forall i > t$. Furthermore, $\sum_{j=s+1}^{t} L_{t,j}\log(A_j)$ is equivalent to $(A_{tree})_t - (A_{tree})_s$. Then, Eqn.(5) becomes:

$$y_t = C_t(\exp\{(A_{tree})_t\} \circ x_0) + \sum_{s=1}^{t} L_{t,s}\exp\{(A_{tree})_t - (A_{tree})_s\} \circ (C_t B_s u_s)$$

where $\circ$ denotes element-wise multiplication. Then the entire sequence of outputs y can be written as:

$$y = STree\_SSM(L, A, B, C)(x_0, u) = M_x x_0 + M_u u$$

where:

$$\begin{aligned} (M_x)_i &= C_i \times diag(\exp\{(A_{tree})_i\}) \\ (M_u)_{ij} &= L_{ij}C_i B_j \times diag(\exp\{(A_{tree})_i - (A_{tree})_j\}) \end{aligned}$$

We note that our method is a generalization of the matrix transformation form of SSM in Mamba2 [6]. When L is a lower triangular causal attention mask, our method is the same as [6] with a non-zero initial state. We introduced the quantity A_{tree}, which accumulates the state transition matrices A according to the tree structure, enabling tree decoding with SSM. The computation for A_{tree} is simple, and incorporating it into the original SSM imposes little overhead. We further note that our method is not limited to a tree structure, but can potentially be used with an arbitrary mask to specify the structure, allowing greater flexibility with SSMs.

Implementation. Based on the methods presented in previous paragraphs, we propose an algorithm in Alg. 1 to perform State-space speculative decoding with tree decoding. The algorithm centers on a tree scan kernel that computes the output of a packed prefix tree input and also caches the intermediate activation values (*i.e.,* $A_{i:j}, B_{i:j}, C_{i:j}$):

$$t'_{i:j}, Cache \leftarrow \textsc{TreeScan}(L, t_{i:j}, x_i)$$

The kernel is hardware-aware as it avoids instantiating any intermediate results as well as the SSM states off from the fast GPU shared memory. During the verification process, we only generate the output $y_{i:j}$ for the input sequence, but not the state after the input. This is because, for speculative decoding, the state at the end of an input sequence is most likely incorrect unless all tokens in the input sequence are accepted. We use the activation replay method [25] to recompute the correct state before the first rejected tokens at the start of the next iteration.

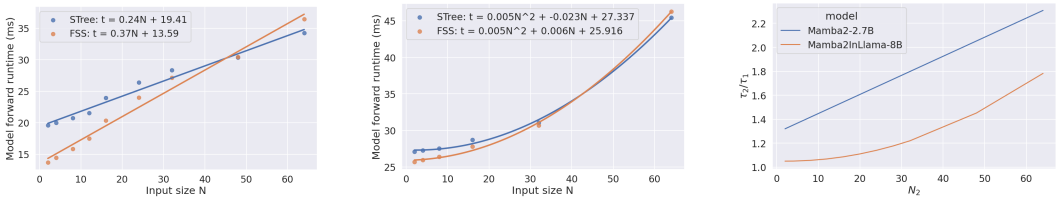
4 Analysis

The wall time of speculative decoding is jointly determined by the runtime of the target model t and the average acceptance length τ. Specifically, we have

$$\text{Wall Time} \propto \frac{t}{\tau}$$

Algorithm 1 Speculative decoding with Tree Scan for SSMs

function SPECULATIVEDECODINGWITHTREESCAN
  Initialize L^* : mask to indicate last accepted token
  Initialize $Cache$: activation cache to facilitate recomputation of state
  Initialize x^* : the correct state
  while $should\_continue$ **do**
    $L_{i:j}, t_{i:j} \leftarrow \text{DRAFT}(t_{i-1})$ ▷ Draft a tree with last accepted token
    $x^* \leftarrow \text{ACTIVATIONREPLAY}(L^*, Cache)$ ▷ Recompute state up to the rejected tokens
    $t'_{i:j}, Cache \leftarrow \text{TREESCAN}(L, t_{i:j}, x^*)$ ▷ Getting output and cache from target model
    $L^*, t_{i:k} \leftarrow \text{FIRSTREJECTED}(t'_{i:j}, t_{i:j})$ ▷ Accept/Reject Drafted tokens
  end while
end function



(a) (b) (c)

Figure 2: **Left**: The runtime for a call to Mamba2-2.7B model vs. input size with STree and Fuse Selective Scan (FSS). A linear regression is performed to obtain the slope and intercept. **Middle**: The runtime for a call to MambaInLlama-8B model vs. input size with STree and FSS. A polynomial regression with degree 2 is used to obtain the parameters. **Right**: The ratio of acceptance length τ required to get wall-clock time improvement vs. input length N_2, with a fixed $N_1 = 5$

The success of speculative decoding relies on the fact that computing an input of length $N > 1$ doesn't increase t too much. As SSMs are not as good as transformers in performing parallel computation due to their recurrent nature, the runtime of the target model calls sees a non-negligible increase even at small input length. Therefore, there exists a trade-off between the gain in acceptance length and the increase in runtime of target model calls as the size of the tree grows. Given that SSM compute and memory cost is linear in the input length, we assume a simple linear model between the model runtime and the input size, where $t = kN + C$, where C is a constant contributed by the time for loading model weights and kN is proportional to input size N, contributed by loading inputs and computation. When the models are large, the constant C dominates the runtime, leading to a negligible effect on runtime by input length N when N is small.

Therefore, to gain wall-clock time improvement we require:

$$\begin{aligned} \frac{k_1 N_1 + C_1}{\tau_1} &\geq \frac{k_2 N_2 + C_2}{\tau_2} \\ \frac{\tau_2}{\tau_1} &\geq \frac{k_2 N_2 + C_2}{k_1 N_1 + C_1} \end{aligned} \tag{7}$$

We note that when applying the above analysis to hybrid models, one key difference is that the transformer blocks in the hybrid models have quadratic space and time complexity in input length, and therefore would require a quadratic model $t = aN^2 + bN + c$ for modeling the runtime and input size, which leads to:

$$\frac{\tau_2}{\tau_1} \geq \frac{a_2 N_2^2 + b_2 N_2 + c_2}{a_1 N_1^2 + b_1 N_1 + c_1} \tag{8}$$

To determine whether we should use tree scan for speculative decoding, we want to evaluate whether the gain in acceptance length can outweigh the increase in runtime. For example, we measured the runtime for Mamba2-2.7B and MambaInLlama-8B model calls using both STree and Fused Selective Scan (FSS), which is an algorithm optimized for short sequence inference [25], with different input sizes in Fig. 2a and Fig. 2b. We performed linear and quadratic regression to obtain an approximation

Table 1: Latency of a Mamba2 forward pass using STree vs. autoregressive forward with selective scan. Note that for 7B, 13B, and 23B models, we initialize the model with random weight to measure the runtime since there is no pretrained model of those sizes.

	2.3B	7B	13B	23B
Autoregressive	10.95	22.94	40.69	72.41
STree	22.36 (2.04x)	33.79 (1.47x)	55.90 (1.37x)	91.08 (1.26x)
Vanilla SD	15.05 (1.37x)	26.46 (1.15x)	45.95 (1.13x)	76.74 (1.06x)

of the parameters (k, C, a, b, c). Assuming that we are comparing STree against vanilla state-space speculative decoding that uses FSS to compute forward pass where we draft 1 sequence with 5 tokens, we can let $N_1 = 5$ and plot the ratio of acceptance length τ_2/τ_1 required to achieve wall-clock time improvement for both models in Fig. 2c. We can see that for STree with Mamba2-2.7B, to gain a wall-clock time improvement at a tree size N_2 of 15 tokens, we need the drafted tree to achieve at least $1.5\times$ acceptance rate as compared to vanilla speculative decoding. On the other hand, STree with Mamba2InLlama-8B requires at least $1.1\times$ acceptance rate with the same tree size, which is much easier to achieve than Mamba2-2.7B. This means that the performance gain that can be achieved with tree decoding with Mamba2InLlama-8B is much more than that with Mamba2-2.7B. This could be due to Mamba2InLlama-8B being a larger model with a larger constant overhead for model call.

Models of bigger size Theoretically, the effectiveness of speculative decoding depends on the runtime difference between the draft model and the target model as well as the acceptance rate of the drafted tokens. Everything else being equal, using a bigger target model leads to a bigger runtime difference. Therefore, the effectiveness of our method on bigger models should hold or improve. In Table 1, we provide the results of latency of a Mamba2 forward pass using STree vs. autoregressive forward with selective scan with same input tree configuration. Note that for 7B, 13B, and 23B models, we initialize the model with random weight to measure the runtime since there is no pretrained model of those sizes. We can see that as the model size grows, the relative overhead of using STree decreases from 2.04x to 1.26x, signaling that as model sizes increase, our method is likely going to be even faster, given that the average acceptance length stays the same. Meanwhile, the gap between the Vanilla SD and STree is closing as model size increases (from 48.9% with 2.3B to 18.9% with 23B), which is another signal that our method is scalable to larger models.

5 Experimental Results

In this section, we aim to demonstrate the efficiency of STree on speculative decoding. In Sec. 5.1, we compare STree against the unrolled input baseline to show that STree is more efficient at decoding a tree with SSMs. Then, in Sec. 5.2, we compare against speculative decoding without tree-based verification to show that STree is able to achieve speed improvement with baseline drafting models and tree construction methods. All experiments are run on an Nvidia RTX 3090 GPU.

5.1 Efficiency of STree against unrolled baseline

Forward pass runtime. Since there is no efficient algorithm currently available for computing tree decoding with SSMs, we evaluate STree against the baseline method of unrolling the token tree into different sequences and computing the output of these sequences using the currently available Fused Selective Scan (FSS) [25], and Chunk Scan [6] kernels. We measure the runtime of a forward pass through a Mamba2-2.7B model. For the input token tree, we use full binary trees of 4/5/6 layers deep, which contain 15/31/63 tokens in the tree respectively. The results are shown in Fig. 3 Right.

STree performs on par with FSS at small tree size (4-layer), as FSS is optimized for short sequences. However, as the size of the tree increases, the forward pass with STree increases slightly from 27.6ms to 34.0ms, while FSS increases drastically from 27.3ms to 59.8ms. This is due to unrolling a prefix tree introduced repeated tokens and extra states as shown in Fig. 3 Left. This slows down the forward pass for both FSS as well as chunk scan. As STree is able to directly decode a packed tree sequence, it demonstrates good scalability as the input tree size grows because it avoids repeated computations.

Tree depth	Methods	#. of States	Tokens Computed
4	Packed	1	15
	Unrolled	8	32
5	Packed	1	31
	Unrolled	16	90
6	Packed	1	63
	Unrolled	32	192

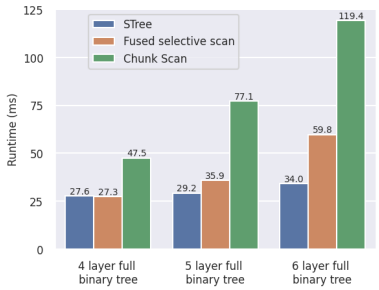


Figure 3: **Left**: Comparison of number of states required and number of tokens computed to get an output for a full binary tree in one forward pass. STree is able to decode a packed tree sequence, while other methods need to unroll the tree into multiple sequences. **Right**: Runtime in milliseconds (ms) for a forward pass for different full binary trees using different algorithms.

Table 2: Generation speed (tokens/second) and memory usage (GB) using STree on packed tree input and Fused Selective Scan (FSS) on unrolled tree input. M is the number of beams we keep at each step and N is the number of steps we beam-searched for. M=4/5 with N=16 for FSS ran Out Of Memory (OOM) on a 3090 GPU with 24GB of GPU memory.

		M=2		**M=3**		**M=4**		**M=5**	
	N	**STree**	**FSS**	**STree**	**FSS**	**STree**	**FSS**	**STree**	**FSS**
Speed	4	**102.69** (1.05×)	97.44	**101.68** (1.12×)	90.64	**94.61** (1.10×)	85.33	**91.80** (1.16×)	78.94
	8	**107.28** (1.09×)	97.53	**104.99** (1.21×)	86.68	**100.67** (1.31×)	76.72	**98.75** (1.44×)	68.44
	16	**105.62** (1.25×)	84.26	**101.97** (1.49×)	68.14	**94.47**	OOM	**88.41**	OOM
Memory	4	**7.67** (0.91×)	8.47	**7.68** (0.85×)	8.97	**7.73** (0.83×)	9.36	**7.75** (0.78×)	9.85
	8	**7.79** (0.86×)	9.05	**7.85** (0.80×)	9.76	**7.88** (0.72×)	10.89	**7.91** (0.64×)	12.32
	16	**8.02** (0.75×)	10.69	**8.12** (0.57×)	14.02	**8.17**	OOM	**8.28**	OOM

Meanwhile, forward pass with chunk scan is consistently slower than both fused selective scan and STree, as it is optimized for parallel training for long sequences and pays a big overhead at a short sequence length [25, 24]. Hence, it should not be considered for use in speculative decoding.

Generation speed against unrolled baseline. Taking one step further, we measure the generation speed of STree with packed tree input against FSS with unrolled tree input. We used a Mamba2-2.7B model as the target model and a Mamba2-130M model as the drafting model. We generate the token tree using beam search [11, 23], where we perform beam search with the drafting model and keep all the tokens generated at each step of beam search, even if the beam is later discarded. This results in an N-layer tree with M tokens at each layer, where N is the number of beam search steps and M is the number of beams. We verify the target model with greedy search, where at each step, the token with the maximum conditional likelihood from the target model is compared to the corresponding child nodes in the token tree to see if we should accept the draft. The acceptance length for the two methods is the same, as the same drafted tree is used for verification. We perform this experiment on the MT_Bench [28] benchmarks for generating 100 tokens. The results are presented in Tab. 2.

We can see that STree with packed input is both faster and more memory efficient than FSS with unrolled input across all tree sizes. We also notice that as the size of the tree grows, the advantage with STree becomes bigger ($1.05\times$ to $1.49\times$ for speed and $0.91\times$ to $0.57\times$ for memory), which coincides with our previous finding that STree forward pass is more scalable to larger trees. Meanwhile, we note that unrolling the tree also adds overhead to the execution. Memory-wise, the increase in memory for STree is almost negligible when tree size increases, mainly due to the increase in input size. On the contrary, the repeated tokens and states required by FSS contribute to a large increase in GPU memory, making a large token tree ($N = 16, M = 4/5$) infeasible to compute.

Table 3: Generation speed (tokens/sec.) and average number of tokens accepted τ for Vanilla speculative decoding (SD) and STreewith MambaInLlama-8B model with 50% mix of transformers.

	MT-Bench		**HumanEval**		**GSM-8K**	
Methods	Speed	τ	Speed	τ	Speed	τ
Temperature=0						
Autoregressive	39.98	1	39.98	1	40.47	1
Vanilla SD	67.68 (1.69×)	2.04	77.18 (1.93×)	2.32	78.38 (1.93×)	2.34
STree	**69.84** (1.74×)	2.47	**78.35** (1.95×)	2.79	**80.03** (1.98×)	2.83
Temperature=1						
Autoregressive	39.72	1	40.04	1	40.16	1
Vanilla SD	49.24 (1.23×)	1.55	52.88 (1.32×)	1.65	51.80 (1.28×)	1.61
STree	**54.08** (1.36×)	2.03	**60.34** (1.50×)	2.26	**58.25** (1.45×)	2.16

5.2 End-to-end generation with hybrid models

Generation with MambaInLlama models. Having verified the effectiveness of STree in performing tree decoding, we then demonstrate the potential of STree in further boosting the speed of generation for SSMs and hybrid models. We choose a Mamba2InLlama-8B[2] [24] hybrid model, with 50% mix of transformers and SSMs, as the target model. As there is a lack of smaller models in the same family, we distill a 2-layer SSM from the target model using data that is used to finetune the target model. We show the results of using a checkpoint from 48000 steps into the distillation. Results from different checkpoints are shown in the ablation study below.

For the vanilla speculative decoding baseline, we use the 2-layer model as the drafting model to draft 1 sequence of 4 tokens every step (target input size 1×5) and verify with the target model output using speculative sampling algorithm [**?**]. For STree, we use the draft model to draft a static tree structure shown in Fig. 4a, and use multi-step speculative sampling (MSS sampling) [19] to verify with the target model output. Both methods use activation replay to backtrack the state. We evaluate the speed of generating 1024 tokens on three different benchmarks: MT_Bench [28], HumanEval [3], and GSM8K [5], with two different temperatures: 0 (greedy) and 1. The results are shown in Tab. 3

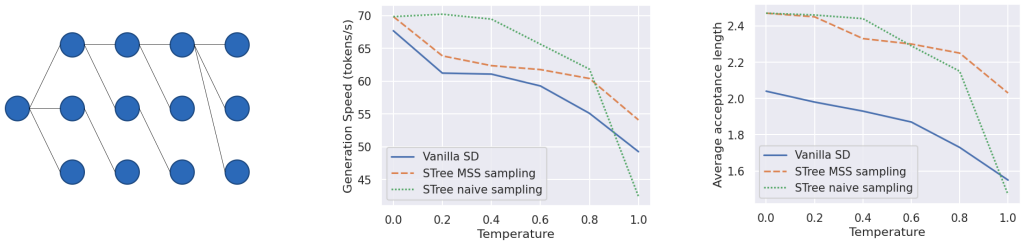
We can see that across all three benchmarks used, STree is able to outperform vanilla speculative decoding. The advantage becomes more obvious when the temperature of the generation increases and the acceptance length drops as a result. We note that this performance improvement is achieved with a baseline tree generation strategy (a static tree) and a draft model still early in its distillation process. With a more advanced tree generation strategy and draft model, the acceptance length with STree will likely improve, which will translate into more generation speed improvement. We include the results with H100 GPUs and a beam search tree in the appendix due to space constraint.

Ablation studies

Effect of temperature We study the effect of sampling temperature on the acceptance rate and speed-up. We use the same setup as the end-to-end generation experiment with MambaInLlama model and vary the temperature used with sampling. From Fig. 4b and Fig. 4c, we can see that average acceptance length drops as temperature goes up for both STree and vanilla speculative decoding, leading to a decrease in generation speed. We notice that the absolute difference between the acceptance length of STree and vanilla speculative decoding is relatively stable (~ 0.4), which means that as the acceptance length drops for both STree and vanilla speculative decoding, the improvement in generation speed becomes bigger, re-affirming our results in the previous sections.

Effect of sampling algorithm Besides multi-step speculative sampling, we test our method with naive sampling used in [19, 23]. Specifically, we use the top-k tokens from the drafting distribution to build our token tree, and perform sampling with the target model. If at any token, the sampled token from the target model falls within the top-k tokens drafted, we accept that token and continue to verify

[2]Checkpoint at https://huggingface.co/JunxiongWang/Llama3.1-Mamba2-8B-distill



(a) (b) (c)

Figure 4: **Left**: Structure of static tree used to generate a prefix token tree with the drafting model. We draft 4 steps for each iteration and 3 tokens for each layer, resulting in 13 tokens in every input sequence. **Middle**: Generation speed of STree and Vanilla Specultaive Decoding (SD) under different temperature. **Right**: Average acceptance length of STree and Vanilla Speculative Decoding (SD) under different temperature.

Table 4: Inference speed (tokens/sec.) and average number of tokens accepted τ for STree with MambaInLlama-8B models with different static tree configurations (sampling temperature = 1).

Static tree configuration	A	B	C	D	E
Max tree width	3	16	6	4	3
Tree depth	4	3	3	4	5
Number of tokens	13	29	16	13	16
Speed	54.08	46.90	51.19	54.39	56.79
τ	2.03	2.15	2.01	2.07	2.09

the next token. We refer the readers to [23] for more details. The results are shown together in Fig. 4b and Fig. 4c. We can see that at low temperatures, naive sampling has a relatively high acceptance length and fast run time. As temperature increases, the acceptance length drops significantly, because the sampled token by the target model is less likely to come from the top-k tokens with the drafted model. This leads to a drop in overall generation speed as well. We note that this drop in generation speed is due to the characteristic of the sampling algorithm, but not our framework. With a high acceptance length at lower temperatures, we can still achieve speed improvements.

Effect of static tree structure We study the effect of different static tree structures on the acceptance rate and speed-up. We note that producing a wider and deeper tree also slows the drafting model and contributes to the overall runtime. The static tree configurations are shown in Fig. 5 in the Appendix. The results are presented in Tab. 4. To determine whether to use a deeper/wider tree, we need to consider not only the effect on the acceptance rate, but also the runtime of the target and drafting models. Increasing the width of the tree drastically does improve the acceptance rate (Column B vs. C), but it also slows down the models and therefore overall speed. When we increase the tree depth (Column A vs. E), we can see that the improvement in τ outweighs the runtime cost, and therefore we see an improvement. Comparing Column A and D, where the number of tokens is the same but with different connectivity, improvement can be seen in both speed and τ when a better tree structure is used. This signifies the importance of having a good tree structure in using STree. This result also agrees with our analysis on the trade-off between runtime and acceptance rate. It provides the promise that when we use a better-trained drafting model and better drafting algorithms, which would give us a better tree and token candidates, the generation speed with STree will improve.

Effect of percentage of transformer blocks in target model Hybrid models are a mix of transformer blocks and SSM blocks. As Transformers have better scaling in parallel compute, we hypothesize that more transformer blocks in the model will lead to a larger improvement in generation speed by STree. We perform an ablation study using Mamba2-Llama3 models [24] with 50%, 25%, and 0% of transformer blocks, using the same drafting model we distilled. The results are shown in Tab. 5. With autoregressive generation, models with more SSM blocks are faster, which agrees with our expectation. For generation with STree, we are still able to outperform vanilla speculative decoding using models with fewer transformer blocks. If we eliminate the effect of different acceptance

Table 5: Inference speed (tokens/sec.) and average number of tokens accepted τ for Vanilla speculative decoding (SD) and STreewith Mamba2-Llama3 models with sampling temperature of 1. The percentage number indicated with the model name is the percentage of transformer blocks in the model. $\frac{\text{Speed-up}}{\tau}$ is obtained by dividing ratio of speed-up against auto-regressive speed in the bracket by τ

	Mamba2-Llama3 (50%)			**Mamba2-Llama3 (25%)**			**Mamba2-Llama3 (0%)**		
Methods	Speed	τ	$\frac{\text{Speed-up}}{\tau}$	Speed	τ	$\frac{\text{Speed-up}}{\tau}$	Speed	τ	$\frac{\text{Speed-up}}{\tau}$
Autoregressive	39.71	1	-	41.65	1	-	43.79	1	-
Vanilla SD	46.56 (1.17×)	1.47	0.80	47.69 (1.14×)	1.45	0.78	53.72 (1.22×)	1.57	0.77
STree	**49.04** (1.23×)	1.84	0.66	**48.08** (1.15×)	1.77	0.64	**55.32** (1.26×)	2.00	0.63

Table 6: End-to-end generation speed (in Tokens/seconds) and acceptance length for Vanilla Speculative decoding and STree using drafting model trained for different steps.

	12000 step	48000 step	264000 step
Speed (Vanilla SD)	**44.52** (1.21x)	53.11 (1.24x)	56.49 (1.42x)
τ (Vanilla SD)	1.34	1.56	1.68
Speed (STree)	42.96 (1.19x)	**58.65** (1.36x)	**64.30** (1.62x)
τ (STree)	1.51	2.02	2.23

lengths in the $\frac{\text{Speed-up}}{\tau}$ column, we can see that models with fewer transformers see less speed-up per acceptance length (0.66 to 0.63), agreeing with our hypothesis.

Effect of different drafting model checkpoints In Table 6, we provide an ablation study on our drafting model distilled to different steps. We can see that as the distillation goes on, the acceptance length for the draft model gets longer. As the acceptance length increases, the end-to-end generation speed of STree improves from being 1% slower than Vanilla SD at step 12000 to 9.67% faster at step 48000 and a further 14.8% speedup at step 264000. This is because the distribution of the drafting model is more aligned with the target model, suggesting that with better drafting model and technique that gives a better acceptance length, the end-to-end runtime of STree will become better.

6 Discussion

Our work proposes STree, an efficient algorithm to unlock the potential of speculative tree decoding for SSMs and hybrid models. We close by discussing the limitations of STree. As presented in our analysis, STree has an overhead at a short input length as compared to previous methods, and will not universally improve generation speed when applied without careful consideration. Reducing this overhead through further algorithmic innovation could lead to a better trade-off. Meanwhile, we only demonstrate improvement on 8B models, which are still considered small LLMs. scaling up our method to bigger models and more advanced speculative decoding methods may bring more benefits, including energy savings for LLM inference. We leave that for the exploration of future work.

Acknowledgements. This work was supported by ONR award #N000142212252 and NSF 2112562 Athena AI Institute.

References

[1] Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas Kohler. Judge decoding: Faster speculative sampling requires going beyond model alignment. *arXiv preprint arXiv:2501.19309*, 2025.

[2] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

[3] Mark Chen et al. Evaluating Large Language Models Trained on Code, July 2021. arXiv:2107.03374 [cs].

[4] Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. SEQUOIA: Scalable and robust speculative decoding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[5] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems, November 2021. arXiv:2110.14168 [cs].

[6] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, pages 10041–10071, 2024.

[7] Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

[8] Paolo Glorioso, Quentin Anthony, Yury Tokpanov, James Whittington, Jonathan Pilault, Adam Ibrahim, and Beren Millidge. Zamba: A compact 7b ssm hybrid model. *arXiv preprint arXiv:2405.16712*, 2024.

[9] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[10] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[11] Wonseok Jeon, Mukul Gagrani, Raghavv Goel, Junyoung Park, Mingu Lee, and Christopher Lott. Recursive speculative decoding: Accelerating llm inference via sampling without replacement. *arXiv preprint arXiv:2402.14160*, 2024.

[12] Arthur J Krener. Bilinear and nonlinear realizations of input-output maps. *SIAM Journal on Control*, 13(4):827–834, 1975.

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

[14] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7421–7432. Association for Computational Linguistics, 2024.

[15] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024.

[16] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

[17] Anders Lindquist and Giorgio Picci. On the stochastic realization problem. *SIAM Journal on Control and Optimization*, 17(3):365–389, 1979.

[18] Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding for accelerating llms via double early exiting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[19] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*, 1(2):4, 2023.

[20] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating generative large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[21] Zongyue Qin, Zifan He, Neha Prakriya, Jason Cong, and Yizhou Sun. Dynamic-width speculative beam decoding for llm inference. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI)*, 2025.

[22] Zongyue Qin, Ziniu Hu, Zifan He, Neha Prakriya, Jason Cong, and Yizhou Sun. Optimized multi-token joint decoding with auxiliary model for llm inference. In *International Conference on Learning Representations (ICLR)*, 2025.

[23] Ruslan Svirschevski, Avner May, Zhuoming Chen, Beidi Chen, Zhihao Jia, and Max Ryabinin. Specexec: Massively parallel speculative decoding for interactive llm inference on consumer devices. *Advances in Neural Information Processing Systems*, 37:16342–16368, 2024.

[24] Junxiong Wang, Daniele Paliotta, Avner May, Alexander M Rush, and Tri Dao. The mamba in the llama: Distilling and accelerating hybrid models. *arXiv preprint arXiv:2408.15237*, 2024.

[25] Yangchao Wu, Yonatan Dukler, Matthew Trager, Alessandro Achille, Wei Xia, and Stefano Soatto. Snakes and ladders: Accelerating ssm inference with speculative decoding. In *NeurIPS Efficient Natural Language and Speech Processing Workshop*, pages 292–304. PMLR, 2024.

[26] Luca Zancato, Alessandro Achille, Giovanni Paolini, Alessandro Chiuso, and Stefano Soatto. Stacked residuals of dynamic layers for time series anomaly detection. *arXiv preprint arXiv:2202.12457*, 2022.

[27] Luca Zancato, Arjun Seshadri, Yonatan Dukler, Aditya Golatkar, Yantao Shen, Benjamin Bowman, Matthew Trager, Alessandro Achille, and Stefano Soatto. B'MOJO: Hybrid state space realizations of foundation models with eidetic and fading memory. *Proc. of NeurIPS*, 2024.

[28] 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. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena, December 2023. arXiv:2306.05685 [cs].

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Claims in the abtract and paper reflects what is done in the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitation is discussed in the Last discussion section.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The methods section contains our algorithms.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Experiment results sections contains these details. Code will also be released after paper acceptance.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code will be open sourced after paper acceptance.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The settings are detailed in the experiment results section.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: **[TODO]**

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Detailed in experiment results section.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: It is briefly discussed in the last discussion section.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: No dataset or new models being proposed.

    Guidelines:

    - The answer NA means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Code will be open sourced after paper acceptance

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification:

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Static Tree configurations

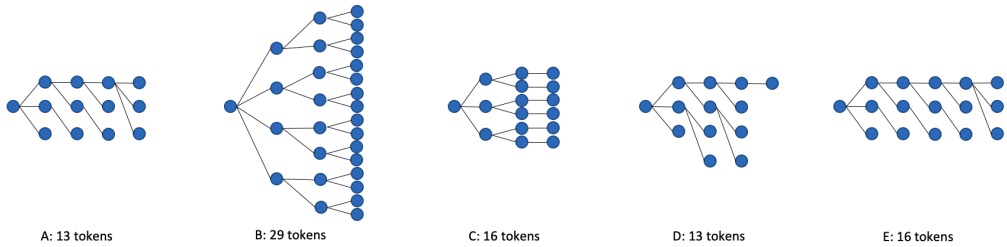


Figure 5: Static tree structure that we used in our ablation study for effect of differnt tree structure.

Fig. 5 show the configurations of the tree we tried in our ablation study. We attempted different breadth (B vs. C), different depth (A vs. E). same number of tokens but different connectivity (A vs. D). The results are show in the

B Experiments on H100 GPU

We further extend our method to H100 GPUs to demonstrate the our method is still applicable with better hardware. We apply our kernels as it is on H100 GPUs and compare it to vanilla speculative decoding.

Table 7: Generation speed (tokens/sec.) and average number of tokens accepted τ for Vanilla speculative decoding (SD) and STree with MambaInLlama-8B model with 50% mix of transformers on H100 GPU.

	MT-Bench		**HumanEval**		**GSM-8K**	
Methods	Speed	τ	Speed	τ	Speed	τ
			Temperature=0			
Autoregressive	78.09	1	78.44	1	79.33	1
Vanilla SD	**113.60** (1.45×)	2.03	**129.76** (1.65×)	2.33	**131.62** (1.66×)	2.33
STree	113.13 (1.45×)	2.45	127.15 (1.62×)	2.77	131.13 (1.65×)	2.81
			Temperature=1			
Autoregressive	76.66	1	77.67	1	78.35	1
Vanilla SD	80.76 (1.05×)	1.56	85.63 (1.10×)	1.64	87.04 (1.11×)	1.63
STree	**85.18** (1.11×)	2.04	**92.74** (1.19×)	2.22	**92.86** (1.19×)	2.16

From the above tables, we can see that the extent of improvement from applying both Vanilla SD and STree is smaller as compared to autoregressive decoding. This is because the memory bandwidth of H100 GPU is bigger than that of RTX3090, leading to a smaller bottleneck in memory transfer. With greedy generation, STree achieves a slightly slower speed as compared to Vanilla SD. When temperature=1, STree still holds an advantage. This is due to the relative increase in acceptance length is smaller when we are doing a greedy generation and the relative overhead between the STree and Vanilla Speculative decoding increases due to the faster GPU. We believe that as the model size increases and the memory bandwidth bottleneck due to the model weight transfer becomes more significant, we will still be able to show improvement. We also note that we present a general algorithm that is not GPU specific and is not optimized for H100 GPU.

C End-to-end generation result using beam search tree

We note that STree can work with any tree generation strategy and optimizing the tree structure is orthogonal to our work. Here, we further show that the end-to-end generation results using a beam search tree for drafting, which is a basic dynamic tree structure. The expansion of the tree depends on the joint probability and will vary from sample to sample. We use greedy decoding with M = 3 and N

Table 8: End-to-end generation speed and acceptance length (τ) for different method of drafting.

	Autoregressive	Vanilla SD	STree w. static tree	STree w. beam search tree
Speed	39.98	67.68 (1.69x)	69.84 (1.75x)	71.60 (1.79x)
τ	1	2.04	2.47	2.60

= 4 (M, N explained in table 1). The results are shown in Table 8. We can see that with a dynamic tree generation strategy, we are able to get a better acceptance length and thus a faster end-to-end generation speed. We believe that more advance tree construction technique would bring even more benefits.