Mamba Drafters for Speculative Decoding

Daewon Choi [1] **Seunghyuk Oh** [1] **Saket Dingliwal** [2] **Jihoon Tack** [1] **Kyuyoung Kim** [1] **Woomin Song** [1 2 †] **Seojin Kim** [3 ‡] **Insu Han** [1] **Jinwoo Shin** [1] **Aram Galstyan** [2] **Shubham Katiyar** [2] **Sravan Babu Bodapati** [2]

Abstract

Speculative decoding has emerged as a promising approach to accelerating large language model (LLM) generation using a fast drafter while maintaining alignment with the target model's distribution. However, existing approaches face a trade-off: external drafters offer flexibility but can suffer from slower drafting, while self-speculation methods use drafters tailored to the target model but require re-training. In this paper, we introduce novel drafters based on Mamba, a state-of-the-art state space model (SSM), as a solution that combines the best aspects of both approaches. By leveraging the linear structure of SSMs, our approach avoids the quadratic complexity inherent in traditional Transformer-based methods, enabling faster drafting and lower memory usage while maintaining the flexibility to work across different target models. We further enhance efficiency with a novel test-time tree search algorithm for generating high-quality draft candidates. Our empirical evaluation demonstrates that Mamba-based drafters not only outperform existing external drafting methods but are also comparable to state-of-the-art self-speculation approaches while using less memory and maintaining their cross-model adaptability.

1. Introduction

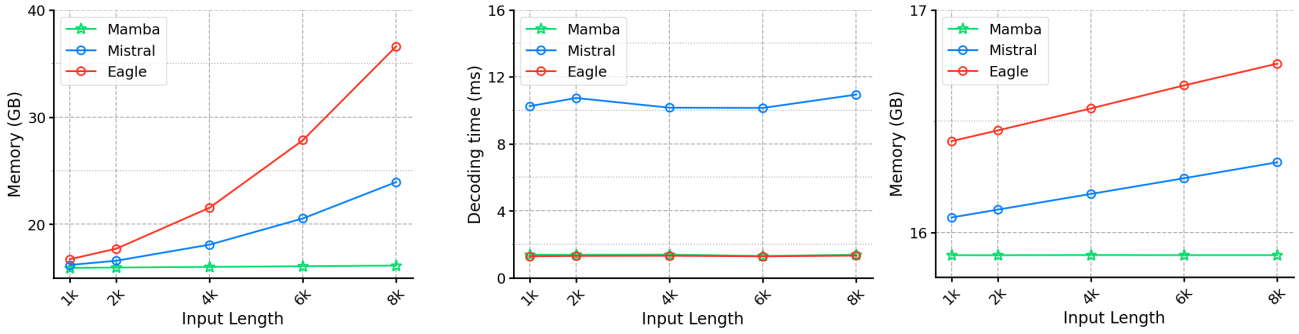
Recent breakthroughs in large language models (LLMs) have been largely driven by Transformer architectures (Vaswani et al., 2017), which have enabled exceptional performance across a wide range of tasks (Achiam et al., 2023; Singhal et al., 2025; Kim et al., 2024a). However, their capabilities often come with significant computational overhead, primarily due to the autoregressive nature of sequential token generation, while the quadratic complexity of the attention mechanism further exacerbates scalability challenges. Speculative decoding (SD) (Stern et al., 2018; Leviathan et al., 2023; Xia et al., 2023; Chen et al., 2023) has emerged as a promising approach to addressing the inefficiencies of autoregressive models by generating multiple candidate tokens with an efficient drafter and verifying them in parallel with the target model, ensuring identical output with greater efficiency. This approach enables simultaneous decoding of multiple tokens within a single forward pass.

Existing SD methods are mainly categorized into two types: (i) using an external drafter (Leviathan et al., 2023) applicable to multiple target models, and (ii) adopting a self-speculation approach, where a drafter is trained to align with the target model itself, showing faster drafting speed compared to external drafters (Cai et al., 2024; Li et al., 2024c). For instance, self-speculation involves training a small Transformer head on top of the target model's large Transformer block to generate multiple candidate tokens, which are then used as a draft for the target model (Li et al., 2024d). While showing a faster drafting speed compared to using an external drafter, training a separate drafter for each target model is computationally expensive. More importantly, to handle distribution shift—i.e., to process novel inputs unseen during training—the drafter should be trained on a large corpus, which is particularly challenging because modifying only the last layer, a common practice, still requires forwarding all lower layers of the large target model during training, leading to significant computational overhead.

This raises a key question: How can we develop an external drafter to have cross-model adaptability while also avoiding the limitation of the Transformer's quadratic computation for fast drafting? This naturally leads us to explore non-quadratic sequential models such as state-space models (SSMs) (Gu et al., 2022b), as external drafters. SSMs leverage a linear recurrence structure with a fixed state size, ensuring per-token computation and memory complexity remain constant during inference, making them more effective drafters than Transformers. Specifically, we use Mamba (Gu & Dao, 2023), a state-of-the-art SSM, as a drafter in SD and make the following key observations:

[†]Work done during an internship at Amazon. [‡]Work done at KAIST. [1]KAIST [2]Amazon AGI [3]Seoul National University. Correspondence to: Sravan Babu Bodapati <sravanb@amazon.com>.

3rd Workshop on Efficient Systems for Foundation Models (ES-FoMo III) at ICML, Vancouver, Canada. 2025.



(a) Encoding Memory (b) Decoding Time (c) Decoding Memory

Figure 1. **Comparison of drafting time & peak memory usage during encoding and decoding.** Mamba drafter maintains nearly constant decoding speed and memory usage, whereas both EAGLE, which employs self-speculation with a single-layer Transformer within the large target model, and the Mistral-based external drafter, which also utilizes a Transformer, exhibit substantially higher memory requirements as the context length increases. Here, the target model is Mistral-7B, and we consider a 160M-sized Mistral and a 130M-sized Mamba. Measurements are taken on an NVIDIA H100.

- *Mamba is an efficient external drafter, showing comparable results with self-speculation*: Mamba's drafting latency is comparable to self-speculation, with both latency and memory usage remaining low even for significantly longer input contexts, in contrast to alternative external drafters.
- *A smaller Mamba can often be more effective than a larger Transformer as an external drafter*: Despite its size, a small Mamba model achieves a comparable acceptance length to larger Transformers, with higher overall throughput due to its fast drafting.

To further leverage Mamba's efficiency, we propose simple yet effective tree decoding strategies with Mamba drafters by formulating decoding as a multi-armed bandit (MAB) (Slivkins et al., 2019) problem. Specifically, we introduce a test-time tree search algorithm that dynamically optimizes the draft tree structure based on the input query. With Mamba's low latency, we observe that it benefits from trees of varying widths and lengths (see Table 6). By framing the selection of the optimal tree structure as an MAB problem, we enable stable and adaptive adjustments of the drafting tree to accommodate different query types.

We conduct a comprehensive set of experiments to evaluate our Mamba-based approach, focusing on practical SD scenarios across a wide range of tasks (Narayan et al., 2018; Zheng et al., 2023; Chen et al., 2021). Our results demonstrate that Mamba-based drafting can significantly outperform traditional Transformer-based approaches. For example, our approach surpasses their throughput by 2x while having similar acceptance length. Furthermore, our approach achieves throughput comparable to EAGLE-2 (Li et al., 2024d), a recent single-layer Transformer drafter designed for a specific target model, in long-context scenarios while consuming up to 20 GB less memory. This is notable as our target-agnostic Mamba drafter works with an arbitrary target model without re-training, whereas EAGLE, a self-speculation method, requires re-training of the drafter whenever the target model is updated. Moreover, advances in SSMs, e.g., Mamba-2 (Dao & Gu, 2024), directly benefit our approach, further enhancing the advantages of using an effective, target-agnostic drafter.

2. Why Mamba for Speculative Decoding?

In this section, we demonstrate that Mamba can serve as powerful external drafters for speculative decoding (SD). We examine this in terms of both efficiency and effectiveness. For *efficiency*, we compare latency and peak memory usage during drafting, which includes encoding (prefill), initial forwarding of the given input sequence, and per-token generation during decoding (in Figure 1). For *effectiveness*, we report throughput, the average number of tokens per unit time, and acceptance length, the average number of tokens accepted per forward pass of the target model (in Figure 2a). To evaluate the efficiency of Mamba as the drafter, we compare it against two baselines: an external Transformer drafter and (i.e., Mistral) a self-speculation drafter (i.e., EAGLE). All drafters are trained from scratch, with additional training details provided in Appendix A.5. All models are instruction-tuned.

2.1. Efficiency of Mamba as a drafter

As Figure 1a illustrates, Mamba offers significant drafting efficiency compared to both baselines. For example, as the input length increases, prefill memory for both Mistral and EAGLE grow nearly quadratically, while Mamba maintains memory usage with its efficient selectivity algorithm (Gu &

Dao, 2023). Furthermore, as in Figure 1b, Mamba exhibits significantly lower decoding latency than Mistral, which is of similar size and is even comparable to EAGLE. Lastly, Figure 1c shows that Mamba's use of a single state enables it to maintain constant memory usage independent of the input length. In contrast, other drafters require the KV cache for decoding, causing the cache size to grow linearly with the input length, leading to high memory overhead. These results demonstrate that Mamba drafters can make SD more adaptable to varying input lengths more effectively than Transformer drafters.

2.2. Effectiveness of Mamba as a drafter

We found that smaller Mamba can be a stronger drafter than larger Mambas, as it shows significant drafting speed, generating more candidates with only slightly lower acceptance rate than larger models (see Figure 2a). Notably, small Mamba achieves a higher acceptance length than the Transformer of similar size. This can be attributed to its better alignment with the distribution of the larger Transformer model, as illustrated in Figure 2b. We believe this highlights the exceptional drafting speed of Mamba, where such a phenomenon is not observed in Transformer-based drafters (see full results in Table 1).

3. Tree-Structured Drafting with Mamba

In this section, we introduce an effective drafting strategy for Mamba drafters by using *tree-structured decoding*, i.e., hierarchically expanding multiple candidate nodes at each step instead of sequentially generating tokens. Specifically, we suggest an efficient way to implement tree search for Mamba decoding (in Section 3.1) and introduce a test-time tree searching algorithm to adaptively optimize draft tree structure (in Section 3.2).

3.1. Tree-structured drafting with Mamba

To improve the effectiveness of the drafter M_q, previous approaches (Yang et al., 2024; Li et al., 2024c) sample multiple candidates from q_i at each drafting step i by constructing a draft tree. Especially in Transformer drafter, this process is accelerated by tree attention (Miao et al., 2024), a specialized attention algorithm that represents the causal relationships between all tokens in the tree, thereby eliminating overlapped token forwarding, e.g., input sequence x_{prefix}. Here, we suggest an efficient tree-structured drafting specialized for Mamba.

Efficient tree-structured drafting with batch generation. We demonstrate that Mamba can perform tree-structured drafting efficiently by using batch generation. Specifically, as Mamba only requires the current state to predict the next token (as it is a recurrent network), generating multiple nodes from the current node only requires copying the current state and then performing sampling. In contrast, Transformers are required to copy the current sequence length of the KV cache to predict the next token, and this overhead grows with the sequence length, making it crucial to eliminate such duplication using tree attention.

Formally, given a tree configuration $\mathcal{T} = (N_1, N_2, ..., N_\gamma)$, where γ is the draft length, and N_i can be understood as the number of new nodes obtained by sampling from each node at the i^{th} generation, one can view tree-structured drafting as a batch generation of a total batch size $\mathcal{B}_i = N_1 \times N_2 \times \cdots \times N_i$.

Efficient cache utilization for batch generation. While efficient, batch generation indeed increases the computation complexity by a factor of $\mathcal{B}_i$ per generation, compared to sequential drafting, i.e., $N_1 = ... = N_\gamma = 1$. To alleviate overheads from the batch size $\mathcal{B}$ during tree-structured drafting for Mamba, we propose a batch-wise cache implementation for Mamba. Specifically, given a tree configuration $\mathcal{T}$, we determine the possible batch sizes $\mathcal{B}_1, ..., \mathcal{B}_\gamma$ for each drafting positions. Using these calculated sizes, we create a state cache per each batch size and allocate memory in advance, preventing the memory re-allocation during duplication. Next, we leverage a graph cache (Nguyen et al., 2021) to accelerate the GPU computation flow for each batch size. This cache stores the graph structure of intermediate computations for each batch size, enabling efficient reuse of the computational graph across multiple executions. The reason this is feasible is that Mamba receives a fixed size of input $(\mathcal{B}_1, 1), (\mathcal{B}_2, 1), ..., (\mathcal{B}_\gamma, 1)$, owing to its linear recurrence structure.

3.2. Test-time dynamic tree search using MAB

We now present a way to systematically allocate the given budget to find the effective tree configuration $\mathcal{T}$, based on the observation that Mamba benefits from different tree configurations across tasks (see Appendix C.6 for details). To this end, we formalize the tree configuration search problem as a multi-armed bandit (MAB) problem and dynamically optimize the tree configuration at inference time.

Decoding as multi-armed bandit. Following a previous work (Kim et al., 2024b), we define each drafting and verification step as a round in the multi-armed bandit (MAB) framework. Specifically, in each round t, drafter M_q follows an policy π that choose k^{th} tree configuration $\mathcal{T}_k^{(t)}$ from the pre-defined tree configuration set $\mathcal{S} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_K\}$. Then it performs γ generations and obtains a reward $r^{(t)}$, e.g., the number of accepted tokens. The goal of the MAB problem is to design an optimal policy π^* that maximizes the expected cumulative reward $\mathbb{E}\left[\sum_{t=1}^{T} r^{(t)}\right]$ over a total

Table 1. **Comparison with Transformer-based external drafters and self-speculation.** We evaluate SD using pre-trained models with both greedy decoding (temperature = 0) and sampling (temperature = 1). All drafters leverage tree-structured drafting and our method additionally uses the proposed tree search algorithm. Throughput is reported along with the acceptance length shown in parentheses. The best results are shown in **bold**.

Target	Drafter		Greedy (Temp=0)			Sampling (Temp=1)		
	Method	Size	XSum	CNN-DM	GSM-8k	XSum	CNN-DM	GSM-8k
Pythia-6.9B	No drafter	–	53.30	49.29	54.69	52.51	45.33	53.81
	Pythia	70M	47.31 (1.52)	46.99 (1.54)	57.36 (1.68)	41.86 (1.67)	45.30 (1.76)	47.96 (1.77)
		160M	50.05 (2.23)	49.53 (2.26)	67.89 (2.72)	46.67 (2.28)	47.17 (2.30)	55.40 (2.63)
		410M	70.53 (4.62)	70.08 (4.73)	75.97 (4.64)	53.50 (3.60)	56.64 (3.80)	63.64 (4.01)
	Ours	130M	**138.80** (4.55)	**131.97** (4.38)	**149.46** (4.57)	**108.68** (3.53)	**105.01** (3.53)	**119.67** (3.73)
Mistral-7B	No drafter	–	51.15	49.55	50.31	53.49	47.40	52.92
	Mistral	160M	61.55 (3.13)	61.04 (3.05)	49.38 (2.21)	53.91 (2.74)	50.50 (2.68)	62.29 (2.94)
	Ours	130M	**76.71** (2.39)	**65.23** (2.13)	**77.50** (2.25)	**79.18** (2.73)	**70.95** (2.65)	**82.63** (2.73)

of T rounds, where T is determined by the completion of generation for a given query.

Optimization. To balance exploration and stable convergence in MAB, we utilize the UCB algorithm (Auer, 2002) as our policy π. It chooses tree configuration $\mathcal{T}_{k^*}^{(t)}$ at round t as follows:

$$k^* = \underset{k\in\{1,..,K\}}{\arg\max}\ \hat{r}_k^{(t)} + \lambda_{\text{UCB}}\sqrt{\frac{2\ln t}{n_k^{(t)}}}, \tag{1}$$

where $\hat{r}_k^{(t)}$ is a cumulative reward mean, i.e., $\sum_{t=1}^{t} r_k^{(t)}$ and $n_k^{(t)}$ is the count numbers of k^{th} configuration is selected up to round t. For reward $r_k^{(t)}$, we define it as follows:

$$r_k^{(t)} := -\left(\frac{1}{N_{\text{accept}}} + \lambda_\gamma \frac{\gamma^{(\mathcal{T}_k)}}{N_{\text{accept}}}\right)\cdot I, \tag{2}$$

where I is an indicator function, which is 1 when the k^{th} configuration is selected and 0 otherwise, and N_{accept} is the number of accepted tokens at round t. Especially, $\gamma^{(\mathcal{T}_k^{(t)})}$ represent draft length of selected tree $\mathcal{T}_k^{(t)}$ to penalize increase of draft times, as γ is increase. We notice this reward directly originated from the SD speedup objective (see Appendix A.6 for more details).

4. Experiments

In this section, we present a comprehensive evaluation of our proposed Mamba drafter framework. Specifically, we evaluate the performance of the pre-trained Mamba drafter across various language modeling tasks. We evaluate different drafter models on XSum (Narayan et al., 2018) and CNN-DailyMail (Hermann et al., 2015) for general language modeling tasks, as well as GSM-8K (Cobbe et al., 2021) for mathematical language modeling. As summarized in Table 1, while larger drafters like Pythia-410M achieve slightly better throughput gains on some datasets due to increased acceptance length, small Transformer drafters (e.g., Pythia-70M) show minimal improvement over the vanilla autoregressive baseline without SD, with only marginal benefits with non-zero temperature. In contrast, Mamba significantly improves throughput across datasets and temperature settings. For instance, in GSM-8K, Mamba achieves nearly 2x the throughput of Pythia-410M on sampling setup. Notably, even when the Mamba drafter has a lower acceptance length than Transformer drafters, it still outperforms them due to its fast drafting speed.

We provide more extensive evaluations and analysis in Appendix C, including evaluations on instruction-tuned models, comparison with self-speculation approaches, long context evaluations, cross-target model performance, and ablations.

5. Conclusion

In this work, we present Mamba-based drafters as an effective solution to the challenges of existing speculative decoding methods. Leveraging the linear structure of state-space models, Mamba significantly improves drafting speed and memory efficiency. To further enhance drafting quality, we introduce a novel tree-based search algorithm. Our experimental results show that Mamba-based drafters not

only outperform existing external drafting techniques but also match the performance of advanced self-speculation approaches, particularly in long-context scenarios.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Auer, P. Finite-time analysis of the multiarmed bandit problem, 2002.

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., Dong, Y., Tang, J., and Li, J. Longbench: A bilingual, multitask benchmark for long context understanding, 2023.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dao, T. and Gu, A. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning (ICML)*, 2024.

Elman, J. L. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Fu, Y., Bailis, P., Stoica, I., and Zhang, H. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Gu, A., Goel, K., Gupta, A., and Ré, C. On the parameterization and initialization of diagonal state space models. In *Advances in Neural Information Processing Systems*, 2022a.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022b.

He, Z., Zhong, Z., Cai, T., Lee, J. D., and He, D. Rest: Retrieval-based speculative decoding. *arXiv preprint arXiv:2311.08252*, 2023.

Hermann, K. M., Kociský, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. Teaching machines to read and comprehend. In *NIPS*, pp. 1693–1701, 2015. URL http://papers.nips.cc/paper/5945-teaching-machines-to-read-and-comprehend.

Kalman, R. E. A new approach to linear filtering and prediction problems. 1960.

Kim, A., Muhn, M., and Nikolaev, V. Financial statement analysis with large language models. *arXiv preprint arXiv:2407.17866*, 2024a.

Kim, T., Jung, H., and Yun, S.-Y. A unified framework for speculative decoding with multiple drafters as a bandit. In *Proceedings of the Fourth Workshop on Efficient Natural Language and Speech Processing (ENLSP-IV): Highlighting New Architectures for Future Foundation Models*, 2024b.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Li, K., Li, X., Wang, Y., He, Y., Wang, Y., Wang, L., and Qiao, Y. Videomamba: State space model for efficient video understanding. *arXiv preprint arXiv:2403.06977*, 2024a.

Li, S., Singh, H., and Grover, A. Mamba-nd: Selective state space modeling for multi-dimensional data. *arXiv preprint arXiv:2402.05892*, 2024b.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024c.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*, 2024d.

Mehta, H., Gupta, A., Cutkosky, A., and Neyshabur, B. Long range language modeling via gated state spaces. In *International Conference on Learning Representations*, 2023.

Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Wang, Z., Zhang, Z., Wong, R. Y. Y., Zhu, A., Yang, L., Shi, X., et al. Specinfer: Accelerating 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, Volume 3*, pp. 932–949, 2024.

Narayan, S., Cohen, S. B., and Lapata, M. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *ArXiv*, abs/1808.08745, 2018.

Nguyen, V., Carilli, M., Eryilmaz, S. B., Singh, V., Lin, M., Gimelshein, N., Desmaison, A., and Yang, E. Accelerating pytorch with cuda graphs, 2021. URL https://pytorch.org/blog/accelerating-pytorch-with-cuda-graphs/. Accessed: 2025-01-27.

Penedo, G., Kydlíček, H., Lozhkov, A., Mitchell, M., Raffel, C., Von Werra, L., Wolf, T., et al. The fineweb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*, 2024.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.

Singhal, K., Tu, T., Gottweis, J., Sayres, R., Wulczyn, E., Amin, M., Hou, L., Clark, K., Pfohl, S. R., Cole-Lewis, H., et al. Toward expert-level medical question answering with large language models. *Nature Medicine*, pp. 1–8, 2025.

Slivkins, A. et al. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

Stern, M., Shazeer, N., and Uszkoreit, J. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama model, 2023.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. Long range arena: A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Wang, J., Paliotta, D., May, A., Rush, A. M., and Dao, T. The mamba in the llama: Distilling and accelerating hybrid models. *arXiv preprint arXiv:2408.15237*, 2024.

Xia, H., Ge, T., Wang, P., Chen, S.-Q., Wei, F., and Sui, Z. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3909–3925, 2023.

Yang, S., Huang, S., Dai, X., and Chen, J. Multi-candidate speculative decoding. *arXiv preprint arXiv:2401.06706*, 2024.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.

Zhu, L., Liao, B., Zhang, Q., Wang, X., Liu, W., and Wang, X. Vision mamba: Efficient visual representation learning with bidirectional state space model. *arXiv preprint arXiv:2401.09417*, 2024.

A. Experimental Details

A.1. Datasets

We evaluate six benchmarks: three for pre-trained models and three for instruction-tuned models. These include XSum (Narayan et al., 2018) and CNN-DailyMail (Hermann et al., 2015) for general language modeling, GSM-8K (Cobbe et al., 2021) for mathematical reasoning, MT-Bench (Zheng et al., 2023) for multi-turn dialogues, Alpaca (Taori et al., 2023) for general instruction-following, and HumanEval (Chen et al., 2021) for code generation. Following EAGLE (Li et al., 2024c), we subsample each dataset to approximately 80 samples.

For evaluating longer-context scenarios, we use six tasks from LongBench (Bai et al., 2023) for document-based question answering: three (NarrativeQA, Qasper, MultifieldQA-en) for Single-Document QA and three (HotpotQA, 2WikiMultihopQA, MuSiQue) for Multi-Document QA. While LongBench originally includes Chinese-language tasks (MultifieldQA-zh, DuReader), we observe that the target model produced poor outputs on these tasks and therefore exclude them. For pre-processing, we first filter out data samples with input lengths exceeding 8k tokens and truncate them to specific input lengths, such as 1k, 2k, 4k, and 8k.

A.2. Baselines.

We use a wide range of Transformer-based drafters as baselines, applying tree drafting for a fair comparison with our Mamba drafter. Additionally, we consider EAGLE-2 (Li et al., 2024d), a recent single-layer Transformer drafter that leverages tree drafting and is directly trained to align with the target model, as a baseline for self-speculation methods.

A.3. Evaluation metrics.

To compare the gains in decoding acceleration from SD across different drafter types, we focus on throughput, which is the number of tokens generated per second during inference, as a measure of overall inference speed. For a more comprehensive evaluation of effectiveness, we also report the average acceptance length, which indicates the average number of tokens accepted per forward pass of the target model.

A.4. Architectures

Pre-train models. For the pre-trained target model, we consider EleutherAI/pythia-6.9b and mistralai/Mistral-7B-v0.1. For external Transformer drafters, we use smaller models from the same family as the target model, specifically Pythia-70M, 160M, 410M and Mistral-160M. For Mamba drafters, we use two versions of Mamba-130M: Mamba-Pythia-130M and Mamba-Mistral-130M, which share tokenizers with Pythia and Mistral, respectively. In cases where no official pre-trained checkpoints are available, e.g., Mistral-160M and Mamba-Mistral, we pre-train them from scratch (see Appendix A.5 for details).

Instruction-tuned models. For the instruction-tuned target model, we consider allenai/open-instruct-pythia-6.9b-tulu and mistralai/Mistral-7B-Instruct-v0.1, which are instruction-tuned from EleutherAI/Pythia-6.9b and mistralai/Mistral-7B-v0.1, respectively. To obtain instruction-tuned drafters, we supervised fine-tune (SFT) the pre-trained drafters on ShareGPT, following the training dataset used for EAGLE (see Appendix A.5 for details). Additionally, we obtain corresponding EAGLE for Pythia and Mistral by following their official released training code.

A.5. Training Details.

Following a common pre-training recipe [1], we pre-train Mistral-160M and Mamba-Mistral-130M on FineWeb-Edu (Penedo et al., 2024) for 5,000 training steps using a batch size of 4,096 and a context limit of 2k. Next, we fine-tune (SFT) the pre-trained drafters on ShareGPT for 2 epochs with a batch size of 128 and a context limit of 2k, following the standard SFT procedure [2]. For validation, we use HellaSwag (Zellers et al., 2019), ARC-Easy (Clark et al., 2018), and PIQA (Bisk et al., 2020). We select the best model by testing various learning rates, specifically {2e-3, 2e-4, 2e-5}.

[1] https://github.com/facebookresearch/lingua

[2] https://github.com/huggingface/alignment-handbook/blob/main/recipes/zephyr-7b-beta/README.md

A.6. Implementation

Tree-structured drafting. Following previous work (Yang et al., 2024), we implement tree-structured drafting for the external Transformer drafter. We use a tree configuration with a depth of 5, i.e., (3,2,2,1,1), to align the draft length with EAGLE-2. For EAGLE-2, we directly follow its official tree-structured drafting implementation.

Reward modeling for MAB. To obtain the reward function in Equation (2), we directly use the speed up formula from SD per drafting step. Given a target model and drafter's decoding time, i.e., per-token generation time, as T_{target} and T_{draft}, the total time of SD $T_{\text{total}}^{\text{SD}}$ per drafting step is as follows:

$$T_{\text{total}}^{\text{SD}} = T_{\text{target}}(\gamma) + \gamma \cdot T_{\text{draft}}, \tag{3}$$

where γ is draft length, and $T_{\text{target}}(\gamma)$ is verification time for forwarding γ draft tokens. Then, we compute SD's decoding time $T_{\text{Avg}}^{\text{SD}}$ by dividing the number of accepted tokens N_{accept}, i.e., $T_{\text{Avg}}^{\text{SD}} = \frac{T_{\text{Total}}^{\text{SD}}}{N_{\text{accept}}}$. Finally, the speed up of SD per drafting step is as follows:

$$\text{speedup} = \frac{T_{\text{target}}}{T_{\text{Avg}}^{\text{SD}}} = N_{\text{accept}} \cdot \frac{T_{\text{target}}}{T_{\text{target}}(\gamma) + \gamma \cdot T_{\text{draft}}} \tag{4}$$

Then, the inverse of speedup is as follows:

$$\frac{1}{\text{speedup}} = \frac{1}{N_{\text{accept}}} \cdot \frac{T_{\text{target}}(\gamma)}{T_{\text{target}}} + \frac{\gamma}{N_{\text{accept}}} \cdot \frac{T_{\text{draft}}}{T_{\text{target}}} \tag{5}$$

Generally, we can assume $\frac{T_{\text{target}}(\gamma)}{T_{\text{target}}} \simeq 1$, as γ is not larger value, and $\frac{T_{\text{draft}}}{T_{\text{target}}} \simeq \lambda_\gamma$ as it is constant during drafting. Then, our reward function r is derived as follows:

$$r = \frac{1}{N_{\text{accept}}} + \lambda_\gamma \cdot \frac{\gamma}{N_{\text{accept}}} \tag{6}$$

This formula originated from the inverse of speed up, so we need to minimize this function.

A.7. Greedy Decoding and Sampling

Algorithm 1 outlines the verification process for draft token acceptance using two decoding strategies:

1. **Greedy decoding** (red, lines 7, 15, 22): This method selects tokens deterministically by setting the temperature to zero, effectively forcing the model to choose the most probable token at each step. This is equivalent to using the one-hot version of the target model, $p_{\text{one-hot}}$.

2. **Sampling-based approach** (blue, lines 8, 16, 23): In contrast, this method introduces stochasticity by sampling tokens from the probability distribution given by the target model p. This allows for more diverse outputs.

The verification algorithm works by comparing the probabilities assigned by the target model and the draft model. The acceptance of a draft token $\tilde{x}_t$ depends on the ratio of these probabilities. If the draft token is rejected, a new token is sampled based on either the greedy or sampling-based approach.

A.8. Computational Resources

We conduct most experiments on a single NVIDIA RTX 4090 24GB GPU, except for longer-context experiments in Table 3, where we use a single NVIDIA H100 80GB GPU to efficiently handle input lengths of up to 8k tokens. For pre-training, we leverage 8 NVIDIA H200 141GB GPUs, which takes approximately one day. For instruction-tuning of external drafters and training EAGLE, we use 8 NVIDIA RTX 4090 24GB GPUs, requiring approximately two hours and one day, respectively. Here, we remark that training EAGLE incurs additional computational cost, as it requires extracting hidden states from the training data via forward passes through the target model.

Algorithm 1 Verification Algorithm.

Given target model p, one-hot version of target model $p_{\text{one-hot}}$, and draft model q.
Given input sequence x_{prefix}, and draft sequence $\tilde{x}$ of length γ.
for $t = 1$ to γ **do**
  Sample u from a uniform distribution:
    $u \sim U[0, 1]$
  Get the probability of each model for the draft token $\tilde{x}_t$
    $p_t = p_{\text{one-hot}}(\tilde{x}_t | x_{\text{prefix}}, x_1, \ldots, x_{t-1})$
    $p_t = p(\tilde{x}_t | x_{\text{prefix}}, x_1, \ldots, x_{t-1})$
    $q_t = q(\tilde{x}_t | x_{\text{prefix}}, x_1, \ldots, x_{t-1})$
  if $u < \min\left(1, \frac{p_t}{q_t}\right)$ **then**
    Accept the token $\tilde{x}_t$:
      $x_t \leftarrow \tilde{x}_t$.
  else
    Reject the draft token $\tilde{x}_t$ and sample a new one:
      $x_t \sim p_{\text{one-hot}}(x | x_{\text{prefix}}, x_1, \ldots, x_{t-1})$.
      $x_t \sim (p(x | x_{\text{prefix}}, x_1, \ldots, x_{t-1}) - q(x | x_{\text{prefix}}, x_1, \ldots, x_{t-1}))_+$.
    break
  end if
end for
if all tokens are accepted **then**
  Sample an extra token $x_{\gamma+1}$:
    $x_{\gamma+1} \sim p_{\text{one-hot}}(x | x_{\text{prefix}}, x_1, \ldots, x_\gamma)$.
    $x_{\gamma+1} \sim p(x | x_{\text{prefix}}, x_1, \ldots, x_\gamma)$.
end if
Output the accepted token sequence $x_1, \ldots, x_n$, where n is the accepted token length.

B. Related Works

State-space models (SSMs). State-space models are strong linear models that combine the classical state-space representation (Kalman, 1960) with recurrent networks (Elman, 1990). In contrast to models with quadratic scaling, such as Transformers (Vaswani et al., 2017), which use self-attention and experience increasing computational costs with sequence length, SSMs leverage linear recurrence (Gu et al., 2022b;a; Mehta et al., 2023), enabling more efficient training and inference. This efficiency enables SSMs to excel, particularly in processing long sequences (Tay et al., 2021).

Recent advancements, such as Mamba (Gu & Dao, 2023), leverage hardware-aware algorithms and selection mechanisms to enhance SSMs further. These developments have enabled SSMs to demonstrate effectiveness in complex tasks across diverse domains, including language, audio, and video (Zhu et al., 2024; Li et al., 2024a;b). Building on this foundation, our research aims to utilize Mamba's efficiency for drafting, enabling faster and more effective speculative decoding.

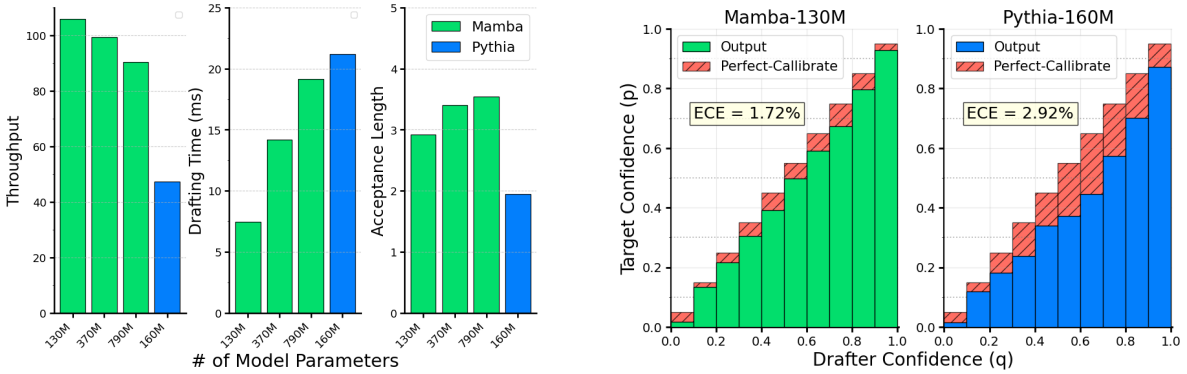
Speculative decoding. Speculative decoding follows a draft-and-verify framework (Stern et al., 2018), where a smaller drafter generates candidate tokens that are verified by the target model. This method accelerates generation by increasing parallelism while ensuring alignment with the target model's distribution. Later advancements extended this approach to sampling settings (Leviathan et al., 2023; Chen et al., 2023) and incorporated various optimization techniques to improve efficiency.

Speculative decoding approaches can be broadly categorized into two types. One approach utilizes external drafter models (Leviathan et al., 2023), which provide high flexibility, allowing a single drafter to be directly used for different target models. On the other hand, self-speculation (Cai et al., 2024; Li et al., 2024c) takes a different approach by utilizing a very small model that uses the target model's internal hidden states for drafting. While being faster, they require expensive re-training of the drafter for every target model, showing limited flexibility. In this work, we demonstrate that Mamba can get the best of both worlds, being an external drafter with very fast drafting speed.

To further improve the acceptance probability of the drafts, recent works move from sequential drafting to tree-structured drafting (Miao et al., 2024; Yang et al., 2024; Li et al., 2024d), which allows verifying multiple draft candidates in parallel. Additionally, researchers have explored non-Transformer drafters (He et al., 2023; Fu et al., 2024). While Wang et al. (2024) introduced hardware optimizations for applying SSMs to speculative decoding, we explored the effectiveness on Transformer-based target models (which is a *de facto* architecture) and developed an efficient inference scheme (i.e., tree-drafting) for Mamba. Furthermore, we present a more thorough comparison and analysis across different drafter models.

C. Additional Results

C.1. Effectiveness of Mamba as a drafter



(a) Comparison of draft efficiency on GSM-8K.

(b) Comparison of draft model calibration.

Figure 2. **Effectiveness of Mamba drafters.** (Left) Mamba drafters achieve substantially higher throughput than a Transformer drafter due to their faster drafting speed and favorable acceptance length. SD is run with a temperature of 1.0 and a draft length of 5. (Right) Reliability diagrams show that a small Mamba drafter aligns better with the target model, Pythia-6.9B, than the Transformer drafter, Pythia-160M, achieving a lower expected calibration error (ECE) on the XSum dataset.

For effective SD, there is an important trade-off between the drafter's speed and size, i.e., a larger drafter may achieve a higher acceptance length than smaller models by generating candidate tokens that are better aligned with the target model's distribution but increase the latency due to the larger size. In this context, we observe that a small Mamba model can be a more effective drafter than a Transformer and, depending on the task, even larger Mamba models. As shown in Figure 2a, the smallest Mamba achieves the highest throughput among all drafters due to its fast drafting and reasonable acceptance length. Notably, the small Mamba achieves a higher acceptance length than the Transformer of similar size. This can be attributed to its better alignment with the distribution of the larger Transformer model, as illustrated in Figure 2b.

C.2. Comparison on instruction-tuned models

Table 2. **Comparison with Transformer-based external drafters and self-speculation.** We evaluate SD using instruction-tuned models with both greedy decoding (temperature = 0) and sampling (temperature = 1). All drafters leverage tree-structured drafting and our method additionally uses the proposed tree search algorithm. Throughput is reported along with the acceptance length shown in parentheses. The best results are shown in **bold**.

	Drafter		Greedy (Temp=0)			Sampling (Temp=1)		
Target	Method	External?	MT-bench	Alpaca	Human-Eval	MT-bench	Alpaca	Human-Eval
Pythia-6.9B	No drafter	–	54.51	55.28	54.76	53.89	54.72	54.21
	Pythia	✓	70.71 (3.10)	60.77 (2.65)	109.51 (4.68)	65.73 (3.03)	62.07 (2.82)	109.52 (4.25)
	EAGLE-2	✗	125.61 (3.85)	**117.17** (3.53)	122.44 (4.71)	87.01 (2.67)	78.58 (2.40)	83.05 (2.97)
	Ours	✓	**128.21** (3.91)	114.08 (3.41)	**172.38** (5.41)	**110.20** (3.65)	**108.54** (3.51)	**143.55** (4.82)
Mistral-7B	No drafter	–	52.97	53.58	52.30	52.39	53.02	52.34
	Mistral	✓	67.47 (3.04)	61.40 (2.73)	100.23 (4.53)	57.19 (2.84)	51.05 (2.40)	80.94 (3.92)
	EAGLE-2	✗	**107.16** (3.22)	94.03 (2.79)	**132.69** (3.98)	**94.03** (2.90)	**86.60** (2.63)	**122.11** (3.78)
	Ours	✓	102.48 (3.16)	**96.83** (2.96)	118.04 (3.69)	88.68 (2.95)	82.75 (2.71)	87.81 (2.94)

Here, we evaluate the Mamba drafter in instruction-following scenarios using MT-bench (Zheng et al., 2023) for multi-turn dialogues, Alpaca (Taori et al., 2023) for general instruction-following tasks, and HumanEval (Chen et al., 2021) for code generation. As shown in Table 2, the Mamba drafter outperforms Transformer drafters across all instruction-following datasets. For example, in HumanEval, while Pythia and Mistral drafters improve throughput over the vanilla autoregressive baseline by 54.75 and 47.93 for their respective target models, Mamba achieves even more significant gains of 117.62 and 65.74 for these models. These results highlight Mamba's flexibility and superior generalization to diverse instruction-following tasks compared to Transformer drafters.

C.3. Comparison with self-speculation

We demonstrate that the Mamba drafter, which is an external drafter, can achieve competitive throughput even against recent approaches that train drafters with direct access to target models. Specifically, we consider EAGLE-2, which uses a single-layer Transformer drafter trained to generate tokens from the target model's last hidden states for better alignment with the target model. Table 2 reports the results on instruction-following tasks, where our Mamba drafter achieves throughput gains comparable to EAGLE-2 across the datasets and target models. Notably, on MT-bench, the Mamba drafter achieves a throughput of 125.61, which is comparable to EAGLE-2's 128.21 for Pythia-6.9B. These results highlight not only Mamba's fast drafting speed (see Figure 1b), but also its effectiveness in achieving comparable acceptance length without requiring access to target models.

Table 3. **Comparisons on LongBench.** Throughput (tokens/s) is the primary metric, with acceptance length shown in parentheses. We also report peak memory, calculated by summing the memory consumption of both the target and drafter models during the prefill phase. Bold indicates the best result, while the runner-up is underlined.

		Single-Document QA				Multi-Document QA				Peak Memory (GB)			
Method	External?	1k	2k	4k	8k	1k	2k	4k	8k	1k	2k	4k	8k
No drafter	-	31.02	27.89	24.35	19.30	28.17	24.22	19.01	14.83	15	16	20	36
Mistral	✓	25.30 (2.43)	23.28 (2.37)	19.48 (2.24)	15.23 (2.21)	24.64 (2.53)	21.06 (2.48)	16.49 (2.44)	12.18 (2.39)	31	33	38	59
EAGLE-2	✗	<u>53.13</u> (2.73)	**47.00** (2.81)	**37.27** (2.76)	**26.12** (2.71)	<u>42.48</u> (2.60)	<u>35.38</u> (2.61)	<u>25.10</u> (2.68)	<u>17.36</u> (2.64)	32	34	42	72
Ours	✓	**55.09** (2.91)	<u>45.65</u> (2.77)	<u>36.32</u> (2.77)	<u>24.92</u> (2.80)	**47.91** (2.94)	**36.40** (2.86)	**26.27** (2.88)	**17.77** (2.87)	31	32	36	52

C.4. Long-context scenarios

To evaluate Mamba's scalability in long-context scenarios, we conduct SD experiments on LongBench (Bai et al., 2023) using input lengths ranging from 1k to 8k, with all drafters trained with the same context limit.[3] Additionally, we apply YaRN (Peng et al., 2023) to extend the context limit for both Transformer drafters and EAGLE-2. As shown in Table 3, which presents the results on LongBench, Mamba maintains a higher acceptance length on longer inputs compared to both Transformer drafters and EAGLE-2, even when the latter utilizes YaRN to extend the context length. In the Single-Document QA task, as the input length increases from 1k to 8k, Mistral's acceptance length decreases from 2.43 to 2.21, while Mamba remains more stable, changing from 2.91 to 2.80. This stability reflects Mamba's ability to extrapolate effectively via recurrence (Gu & Dao, 2023). Moreover, Mamba generalizes well to unseen complex distributions. In the Multi-Document QA task (which is a complex problem compared to single-document QA as the answer is located across multiple documents), Mamba consistently achieves throughput gains comparable to EAGLE-2. Here, we notice that the gains are obtainable more efficiently: when applying to 8k, Mamba only consumes memory up to 52GB, compared to EAGLE-2, which reaches up to 72GB (with both including the memory needed for drafter and target verification).

C.5. Cross-target model performance

Table 4. **Cross-target model performance on MT-bench.** Experiments are run with a temperature of 0 and sequential drafting with a length of 5.

Method	Setup	Accept length	Throughput
EAGLE-2	Pythia → Pythia	2.59	94.75
	Mistral → Pythia	N/A	N/A
Ours	Pythia → Pythia	3.08	112.69
	Mistral → Pythia	2.45	93.20

Using an external drafter enables plug-and-play integration with new target models without the need to re-train the drafter for each specific model. To evaluate Mamba's performance as an external drafter, we use the Mamba drafter with a target model that the drafter has not been explicitly trained to align with. Specifically, we use the instruction-tuned variant of Pythia-6.9B as the target model and Mamba trained with the Mistral-7B tokenizer. As shown in Table 4, even without explicit training, Mamba achieves throughput comparable to EAGLE-2, which is specifically trained for the target model. This highlights Mamba's flexibility as an external drafter, enabling efficient deployment without the need for costly re-training.

C.6. Ablations and analysis

We further evaluate the contributions of individual components in our framework to the gains in decoding acceleration. Here we mainly consider throughput (tokens/sec) as the performance metric.

[3]Following EAGLE-2, we fine-tune pre-trained Mamba and Mistral on ShareGPT with a context limit of 2k.

Tree-structured drafting. In Table 5, we show the impact of tree-structured drafting on performance. Our approach improves acceptance length with minimal drafting latency overhead, resulting in higher throughput gains. This effect is similar to the tree-attention mechanism used in Transformer-based drafters. These results demonstrate that our batch generation enables Mamba to benefit from tree-structured drafting, which has not been explored in the field to the best of our knowledge.

Table 5. **Tree-structured drafting on MT-bench.** Experiments are run with a temperature of 0 and a fixed tree configuration of $(3, 2, 2, 1, 1)$. Tree drafting yields notable improvements in all performance metrics.

Tree?	Accept length	Latency	Throughput
✗	3.08	6.62	112.69
✓	3.91	8.30	127.37

Tree configurations. Table 6 shows that Mamba maintains stable throughput across diverse configurations, whereas Transformer drafters exhibit a decline as tree length increases. This is due to Mamba's very fast drafting, which effectively mitigates the overhead of using longer trees.

Table 6. **Throughput by tree configuration.** Throughput (tokens/s) for different tree configurations given as $(N_1, ..., N_\gamma)$, where N_i indicates the number of samples at i^{th} generation in drafting step.

Method	(3,3,2,1)	(3,2,2,1,1)	(2,2,2,1,1,1)
Pythia	75.38	70.71	63.75
Ours	124.99	127.37	124.37

Test-time tree search. We further analyze the impact of the test-time tree search algorithm. Specifically, we use as tree candidates $(3, 3, 2, 1)$, $(3, 2, 2, 1, 1)$, and $(2, 2, 2, 1, 1, 1)$ and compare them with naive tree-structured drafting that utilizes a fixed tree configuration, i.e., $(3, 2, 2, 1)$. As shown in Table 7, our multi-armed bandit (MAB)-based algorithm often improves throughput significantly on several datasets (e.g., HumanEval) compared to the naive approach.

Table 7. **Effects of test-time tree search on throughput.** Experiments are run with a temperature of 0.

Search?	MT-bench	Alpaca	HumanEval	Avg.
✗	124.99	**116.12**	149.15	130.09
✓	**128.21**	114.08	**172.38**	**138.22**