TreeFlash: Parallel AR-Approximation for Faster Speculative Decoding

Peer Rheinboldt and Frédéric Berdoz and Roger Wattenhofer
ETH Zürich, Switzerland
{prheinboldt,fberdoz,wattenhofer}@ethz.ch

Abstract

One-shot block drafters for speculative decoding generate the full draft in a single forward pass, achieving strong throughput by eliminating sequential token generation. However, they predict each draft token conditioned only on the prefix context, with no dependence on previously drafted tokens. This non-autoregressive conditioning causes the drafter's distribution to diverge from the verifier's true autoregressive distribution as draft depth grows. This problem becomes more severe in tree-based drafting, where distinct branches are forced to share the same marginal distribution for subsequent tokens. We propose **TreeFlash**, which addresses this by incorporating an MLP layer conditioned on the drafter's hidden state and the previous token to approximate an autoregressive distribution. TreeFlash retains the $\mathcal{O}(1)$ decoding time complexity of one-shot drafters by employing a two-stage approximation mechanism. TreeFlash achieves state-of-the-art performance across a variety of tasks and models, improving over marginal tree drafting by 12% higher block efficiency and 9% higher speedup at a drafting budget of 64. [1]

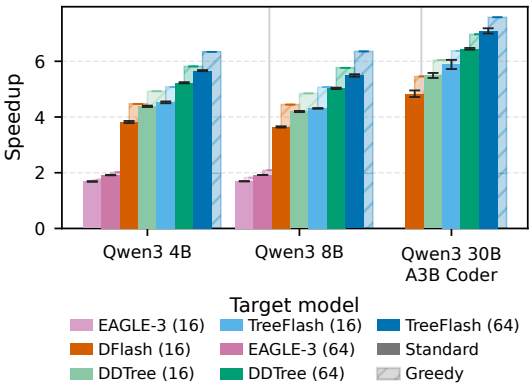


Figure 1: Speedup averaged across datasets for standard (solid) and greedy (translucent) decoding. Draft budget B is reported in parentheses next to each method. TreeFlash consistently outperforms both DFlash ($+17.8\%$) and DDTree ($+4.0\%$) under the same draft budget. Under increased draft budget $B = 64$ TreeFlash achieves $+9.1\%$ speedup over DDTree.

1 Introduction

The autoregressive nature of transformer-based large language models (LLMs) (Vaswani et al., 2017) fundamentally limits their inference throughput, a constraint that grows increasingly acute as frontier models continue to scale (Yang et al., 2025; Singh et al., 2025). A paradigm that escapes the quality–efficiency trade-off of compression-based approaches is *speculative decoding* (Leviathan et al., 2023), in which a lightweight *drafter* proposes a block of γ candidate tokens that are then verified in a single parallel forward pass of the larger *verifier*. This exploits the memory-bandwidth underutilization of standard autoregressive decoding by amortizing costly parameter transfers via batched verification.

A key development in speculative decoding is *tree-based drafting* (Miao et al., 2024; Sun et al., 2023; Wang et al., 2025), in which the drafter generates a tree of candidate continuations rather than a single sequence. By exploring multiple branches simultaneously, tree drafting substantially increases the expected number of accepted tokens per verification step.

A newer frontier in drafting is *one-shot block generation*: rather than producing tokens sequentially, the drafter generates the entire draft block in a single forward pass. DFlash (Chen et al., 2026) shows that a single diffusion-like pass, conditioned on intermediate representations reused from the verifier, achieves substantial speedups over prior autoregressive drafters. Concurrent to our work, DDTree (Ringel and Romano, 2026) applies the OPT-Tree construction algorithm of Wang et al.

[1] Our code is available at github.com/ETH-DISCO/TreeFlash

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 45297–45310
October 24-29, 2026 ©2026 Association for Computational Linguistics

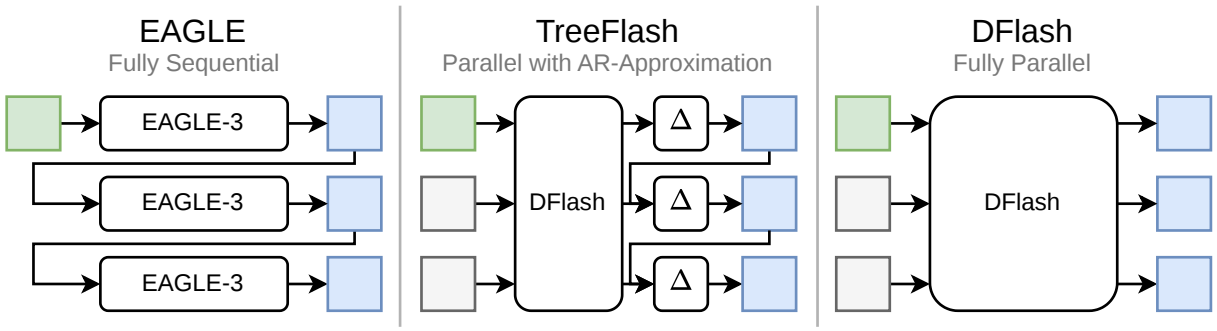


Figure 2: Overview of different drafting paradigms. EAGLE-3 (Li et al., 2025b) (left) is a small autoregressive drafter. DFlash (Chen et al., 2026) (right) is a single-pass parallel drafter. TreeFlash (middle) adapts DFlash by including a lightweight AR-approximator layer that allows fully parallel AR-approximation.

(2025) to DFlash's output distribution, yielding further improvements in acceptance and throughput.

However, these one-shot drafters have a fundamental limitation: the predicted distribution for draft token x_{t+i} is conditioned only on the prefix context $x_{\leq t}$, with no dependence on preceding drafted tokens. This non-autoregressive conditioning causes the drafter's distribution to increasingly diverge from the verifier's true autoregressive distribution as draft depth grows. In tree-based settings, this problem is compounded: distinct branches that share a common prefix are forced to share the same marginal distribution for subsequent tokens, degrading tree quality.

In light of this limitation, we introduce **TreeFlash**, a single-pass drafter which incorporates a lightweight AR-approximation mechanism to overcome the conditioning issue while preserving the fully parallel nature of one-shot drafting.

Our key contributions include:

- We identify the lack of autoregressive conditioning as a key bottleneck of one-shot blockwise drafters, particularly in tree-based settings.
- We introduce a lightweight AR-approximation algorithm that preserves the fully parallel nature of one-shot drafting.
- We show that TreeFlash improves the performance over fully marginal distributions and motivate our design choices through ablations of the architecture and training objective.

2 Related Work

2.1 Speculative Decoding

Speculative decoding is a lossless paradigm for accelerating autoregressive generation in large language models. Early work employs a small, separate language model as a drafter to propose candidate tokens that are subsequently verified by the target model in parallel (Leviathan et al., 2023).

A key advancement in this line of research is tree-based speculative decoding, in which the drafter constructs a tree of candidate continuations rather than a single sequence, substantially improving acceptance rates. Early theoretical treatments derive sophisticated token-acceptance criteria to optimally exploit the drafted tree (Miao et al., 2024; Sun et al., 2023). More recent state-of-the-art methods find that a simple equality-based acceptance check is sufficient in practice, as it admits efficient non-stochastic tree construction strategies (Wang et al., 2025; Li et al., 2024).

Alongside improvements to tree speculative decoding, considerable effort has been directed at reducing the cost of the drafter itself. Whereas early work relies on entirely separate small autoregressive models, more recent approaches replace these with lightweight models that reuse intermediate hidden states from the verifier (Li et al., 2025b). Nevertheless, such models remain fundamentally constrained by autoregressive decoding, limiting the length of the drafted sequence.

To overcome this bottleneck, several works have explored fully parallel draft generation. Medusa and Hydra attach linear prediction heads directly to the verifier backbone to predict future tokens simultaneously (Cai et al., 2024; Ankner et al., 2024). More recently, diffusion models have been proposed as drafters (Li et al., 2025a; Sandler et al., 2025). While diffusion methods eliminate the sequential nature of draft generation, they still require multiple iterations to produce a final draft.

A complementary direction, TiDAR, incorporates parallel drafting directly into the model pre-

training objective, enabling a single model to generate draft tokens in parallel and verify them autoregressively, with no observed degradation in output quality relative to standard autoregressive models (Liu et al., 2025).

2.2 One-Shot Block-Wise Drafters

A recent line of work eliminates multi-step generation entirely, producing the full draft sequence in a single model call. DFlash shows that a single diffusion-like forward pass, conditioned on features reused from the verifier, achieves substantial speedups over prior methods (Chen et al., 2026). Concurrent to our work, DDTree extends this by applying the OPT-Tree construction algorithm of Wang et al. (2025) to DFlash's output distribution, yielding further improvements in acceptance and throughput (Ringel and Romano, 2026). DART constructs a tree from independent positional token distributions, but relies on an external N-gram continuity model and the construction of a large N-gram trie at inference time (Liu et al., 2026).

3 Method

AR-Approximation. A central limitation of block-wise drafters such as DFlash is that they predict only a marginal distribution $q(x_{t+i}|x_{\leq t})$ with no dependence on the immediately preceding tokens. This is problematic for two reasons. First, while coherent drafts can still be produced in the greedy (T=0) single-sequence paradigm, conditioning becomes crucial in the tree-based setting, where multiple parallel candidate sequences are generated. Second, even under the ground-truth marginal distribution $p(x_{t+i}|x_{\leq t})$, the total variation distance (TVD) to the verifier's AR distribution $p(x_{t+i}|x_{<t+i})$ grows substantially with draft depth, as illustrated in Figure 3.

Some form of conditioning on previously drafted tokens within the block drafter is therefore necessary. The key challenge is to incorporate such conditioning without sacrificing the $\mathcal{O}(1)$ decoding time complexity that makes one-shot block drafting attractive. We introduce a lightweight SwiGLU layer that produces a modified hidden state for position $t+i$ by incorporating the input embedding of the preceding token:

$$h'_{t+i} = h_{t+i} + \text{SwiGLU}\left(\tilde{h}_{t+i} :: e_{t+i-1}\right) \quad (1)$$

, where $\tilde{h}_{t+i}$ is the normalized hidden state and e_{t+i-1} is the normalized input embedding of the preceding token. We then compute the modified token distribution $q'(x_{t+i}|x_{\leq t}, x_{t+i-1})$ by applying the verifier's output embedding to h'_{t+i}.

Inference. We follow the OPT-Tree construction algorithm (Wang et al., 2025), which greedily selects a set of B candidate nodes maximizing the expected number of accepted tokens by retaining those paths with the highest product of drafter token probabilities. Throughout the paper, B denotes the *draft budget*, i.e., the total number of candidate nodes submitted for verification per speculative decoding step. Since token selection is deterministic, we use a simple equality check as the verification procedure, ensuring lossless generation.

Naively applying the AR-approximator of Equation 1 requires at least γ sequential forward passes through the AR-approximator, partially offsetting the efficiency gains of one-shot drafting. We avoid this overhead via a two-stage construction. In the first stage, we use the original DFlash distribution to build a top-M-ary tree, where M is a hyperparameter controlling the branching factor (see Section 4.2 for an empirical analysis). In the second stage, the AR-approximator is applied to the nodes of this M-ary tree to obtain modified token distributions, which are then used to construct the final draft tree via OPT-Tree selection. Because the AR-approximator only conditions on the preceding token, and because the M-ary tree contains exactly M unique tokens at each depth, all modified distributions can be computed in $M \cdot \gamma$ parallel evaluations regardless of B, preserving $\mathcal{O}(1)$ drafting time complexity. Note that this construction requires all non-leaf nodes of the final draft tree to lie within the M-ary tree, while leaf nodes may fall outside it. A detailed description of the algorithm is provided in Appendix B.

Training. For each training sample, multiple random anchor positions are sampled at which the drafter gets evaluated. Ground-truth tokens are used as input for the preceding tokens in the AR-approximation, ensuring the model is trained on faithful input-output pairs. Diverging from DFlash, we change the loss from cross-entropy on target tokens to forward KL-divergence to the verifier distribution, which has been shown to yield superior draft quality (Zhou et al., 2024). We initialize the DFlash backbone from the pretrained DFlash checkpoint and zero-initialize the AR-approximator to ensure training starts with a faithful copy of DFlash. In line with DFlash, we adopt

Method	GSM8K Speedup	GSM8K τ	MATH-500 Speedup	MATH-500 τ	HumanEval Speedup	HumanEval τ	MBPP Speedup	MBPP τ	MT-Bench Speedup	MT-Bench τ	Mean Speedup	Mean τ
T=1 (Sampling)												
Target model: Qwen3 4B												
EAGLE3 (16)	$1.79^{0.01}$	$3.15^{0.01}$	$1.68^{0.01}$	$2.93^{0.01}$	$1.70^{0.01}$	$2.96^{0.01}$	$1.64^{0.02}$	$2.88^{0.02}$	$1.62^{0.02}$	$2.84^{0.02}$	$1.69^{0.01}$	$2.95^{0.01}$
DFlash (16)	$4.19^{0.06}$	$5.65^{0.08}$	$4.39^{0.09}$	$5.87^{0.12}$	$4.24^{0.05}$	$5.66^{0.05}$	$3.96^{0.08}$	$5.31^{0.08}$	$2.33^{0.04}$	$3.11^{0.02}$	$3.82^{0.04}$	$5.12^{0.05}$
DDTree (16)	$4.78^{0.08}$	$6.66^{0.10}$	$4.96^{0.08}$	$6.86^{0.08}$	$4.82^{0.07}$	$6.68^{0.11}$	$4.52^{0.10}$	$6.22^{0.17}$	$2.84^{0.02}$	$3.87^{0.02}$	$4.38^{0.02}$	$6.06^{0.05}$
TreeFlash (16)	$\mathbf{4.98}^{0.04}$	$\mathbf{7.15}^{0.07}$	$\mathbf{5.12}^{0.13}$	$\mathbf{7.31}^{0.19}$	$\mathbf{4.99}^{0.08}$	$\mathbf{7.13}^{0.08}$	$\mathbf{4.67}^{0.07}$	$\mathbf{6.73}^{0.09}$	$\mathbf{2.87}^{0.01}$	$\mathbf{4.07}^{0.01}$	$\mathbf{4.53}^{0.03}$	$\mathbf{6.48}^{0.04}$
EAGLE3 (64)	$2.06^{0.01}$	$3.66^{0.03}$	$1.91^{0.02}$	$3.41^{0.01}$	$1.91^{0.02}$	$3.39^{0.01}$	$1.87^{0.01}$	$3.32^{0.02}$	$1.83^{0.00}$	$3.27^{0.00}$	$1.92^{0.01}$	$3.41^{0.01}$
DDTree (64)	$5.71^{0.05}$	$7.74^{0.06}$	$5.88^{0.08}$	$7.92^{0.09}$	$5.81^{0.06}$	$7.82^{0.05}$	$5.39^{0.03}$	$7.29^{0.01}$	$3.35^{0.00}$	$4.50^{0.04}$	$5.23^{0.02}$	$7.05^{0.04}$
TreeFlash (64)	$\mathbf{6.22}^{0.01}$	$\mathbf{8.72}^{0.04}$	$\mathbf{6.34}^{0.09}$	$\mathbf{8.84}^{0.14}$	$\mathbf{6.23}^{0.15}$	$\mathbf{8.67}^{0.21}$	$\mathbf{5.95}^{0.11}$	$\mathbf{8.34}^{0.12}$	$\mathbf{3.58}^{0.04}$	$\mathbf{4.95}^{0.03}$	$\mathbf{5.66}^{0.02}$	$\mathbf{7.90}^{0.02}$
Target model: Qwen3 8B												
EAGLE3 (16)	$1.82^{0.01}$	$3.13^{0.01}$	$1.72^{0.00}$	$2.94^{0.01}$	$1.78^{0.01}$	$3.06^{0.02}$	$1.63^{0.00}$	$2.80^{0.01}$	$1.51^{0.01}$	$2.58^{0.01}$	$1.69^{0.00}$	$2.90^{0.00}$
DFlash (16)	$4.21^{0.06}$	$5.65^{0.07}$	$4.26^{0.06}$	$5.68^{0.05}$	$3.95^{0.05}$	$5.26^{0.06}$	$3.64^{0.04}$	$4.93^{0.04}$	$2.16^{0.05}$	$2.88^{0.01}$	$3.64^{0.02}$	$4.88^{0.02}$
DDTree (16)	$4.77^{0.03}$	$6.62^{0.03}$	$4.84^{0.09}$	$6.66^{0.11}$	$4.46^{0.07}$	$6.16^{0.09}$	$4.25^{0.03}$	$5.91^{0.03}$	$2.67^{0.03}$	$3.64^{0.04}$	$4.20^{0.02}$	$5.80^{0.02}$
TreeFlash (16)	$\mathbf{4.85}^{0.03}$	$\mathbf{7.03}^{0.03}$	$\mathbf{4.88}^{0.07}$	$\mathbf{7.02}^{0.06}$	$\mathbf{4.63}^{0.03}$	$\mathbf{6.63}^{0.04}$	$\mathbf{4.44}^{0.03}$	$\mathbf{6.37}^{0.05}$	$\mathbf{2.75}^{0.01}$	$\mathbf{3.91}^{0.02}$	$\mathbf{4.31}^{0.01}$	$\mathbf{6.19}^{0.01}$
EAGLE3 (64)	$2.06^{0.01}$	$3.63^{0.01}$	$1.94^{0.00}$	$3.41^{0.02}$	$2.03^{0.01}$	$3.55^{0.01}$	$1.86^{0.02}$	$3.25^{0.04}$	$1.71^{0.01}$	$2.98^{0.01}$	$1.92^{0.00}$	$3.36^{0.01}$
DDTree (64)	$5.67^{0.04}$	$7.67^{0.07}$	$5.73^{0.09}$	$7.65^{0.12}$	$5.40^{0.10}$	$7.26^{0.12}$	$5.15^{0.11}$	$6.95^{0.13}$	$3.18^{0.04}$	$4.26^{0.03}$	$5.03^{0.02}$	$6.76^{0.03}$
TreeFlash (64)	$\mathbf{6.21}^{0.12}$	$\mathbf{8.69}^{0.18}$	$\mathbf{6.30}^{0.05}$	$\mathbf{8.70}^{0.04}$	$\mathbf{5.88}^{0.09}$	$\mathbf{8.13}^{0.11}$	$\mathbf{5.66}^{0.06}$	$\mathbf{7.86}^{0.07}$	$\mathbf{3.41}^{0.09}$	$\mathbf{4.69}^{0.04}$	$\mathbf{5.49}^{0.04}$	$\mathbf{7.61}^{0.06}$
T=0 (Greedy)												
Target model: Qwen3 4B												
EAGLE3 (16)	1.84	3.18	1.78	3.09	1.75	3.05	1.69	2.94	1.71	2.97	1.75	3.05
DFlash (16)	4.63	6.26	5.96	7.93	4.87	6.44	4.37	5.83	2.52	3.48	4.47	5.99
DDTree (16)	5.13	7.13	6.13	8.45	5.39	7.43	4.91	6.80	3.07	4.20	4.92	6.80
TreeFlash (16)	**5.29**	**7.62**	**6.34**	**9.05**	**5.50**	**7.83**	**5.05**	**7.27**	**3.20**	**4.55**	**5.08**	**7.27**
EAGLE3 (64)	2.14	3.73	2.06	3.63	2.00	3.47	1.94	3.38	1.96	3.41	2.02	3.53
DDTree (64)	6.07	8.24	7.24	9.75	6.36	8.55	5.85	7.90	3.56	4.87	5.82	7.86
TreeFlash (64)	**6.70**	**9.39**	**7.90**	**10.99**	**6.85**	**9.51**	**6.44**	**8.96**	**3.82**	**5.37**	**6.34**	**8.84**
Target model: Qwen3 8B												
EAGLE3 (16)	1.88	3.21	1.95	3.32	1.87	3.18	1.74	2.97	1.69	2.86	1.82	3.11
DFlash (16)	4.71	6.33	5.96	7.88	4.78	6.34	4.36	5.87	2.42	3.33	4.45	5.95
DDTree (16)	5.15	7.14	6.12	8.42	5.25	7.20	4.68	6.47	3.04	4.16	4.85	6.68
TreeFlash (16)	**5.33**	**7.71**	**6.38**	**9.20**	**5.40**	**7.74**	**5.03**	**7.24**	**3.23**	**4.55**	**5.07**	**7.29**
EAGLE3 (64)	2.17	3.75	2.23	3.81	2.16	3.70	1.96	3.35	1.93	3.30	2.09	3.58
DDTree (64)	6.15	8.25	7.15	9.56	6.22	8.35	5.73	7.68	3.59	4.87	5.77	7.74
TreeFlash (64)	**6.73**	**9.43**	**7.99**	**11.02**	**6.79**	**9.40**	**6.41**	**8.96**	**3.85**	**5.40**	**6.35**	**8.84**

Table 1: Block efficiency (τ) and speedup across datasets and configurations. Numbers in grey denote the tree/draft budget B. Q3-4B and Q3-8B refer to Qwen3 4B and Qwen3 8B respectively; see Table 2 for Qwen3 Coder 30B A3B. For T=1 rows, superscripts report standard deviations across run-level means. TreeFlash consistently outperforms both baselines across all configurations. Under the budget-matched setting (B=16), transitioning from chain to tree drafting (DFlash $\rightarrow$ DDTree) already accounts for 63% of the block-efficiency gain and 73% of the speedup gain that TreeFlash achieves over DFlash, with AR-approximation contributing a further $+0.50$ in τ and $+0.19\times$ in speedup on top. Under the increased-budget setting (B=64), the benefit of AR-approximation grows: TreeFlash improves over DDTree by $+0.96$ in τ ($+12.4\%$) and $+0.52\times$ in speedup ($+9.1\%$). We note that EAGLE-3 was trained on a different dataset and is included for reference rather than as a direct comparison.

the same loss scaling scheme, penalizing earlier tokens more heavily.

4 Experiments

Baselines. We evaluate against three state-of-the-art drafters. *EAGLE-3* (Li et al., 2025b) is a small autoregressive drafter that generates tokens sequentially. We use the checkpoints provided by AngelSlim (Contributors, 2025). It is important to note that these were trained on a different data distribution from DFlash and TreeFlash. *DFlash* (Chen et al., 2026) produces a sequence draft in a single forward pass, and *DDTree* (Ringel and Romano, 2026) applies the OPT-Tree construction algorithm on top of this. DFlash, DDTree, and TreeFlash share the same underlying pretrained checkpoint. TreeFlash additionally finetunes this checkpoint as described in Section 3 and adds extra parameters; see Table 3 for an equivalently finetuned DFlash model. Target models are Qwen3 4B ($+125M$), Qwen3 8B ($+251M$), and Qwen3 Coder 30B A3B ($+63M$) (Yang et al., 2025).

Datasets. We evaluate across a diverse set of tasks. For mathematical reasoning, we use *MATH-500* (Lightman et al., 2024) and *GSM8K* (Cobbe

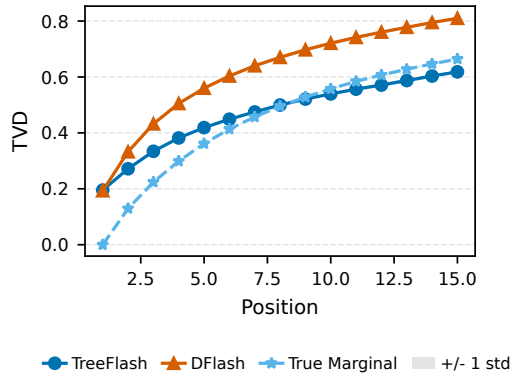


Figure 3: Total Variation Distance (TVD) to the verifier distribution across draft positions for Qwen3 4B. We compare DFlash, TreeFlash, and an approximation of the ground-truth marginal distribution. Both DFlash and TreeFlash have similar TVD at early tokens, however, TreeFlash gets increasingly better as the draft depth increases. TreeFlash surpasses the ground-truth marginal distribution at depths beyond 9, suggesting that AR-approximation provides benefits beyond improving the marginal distribution alone.

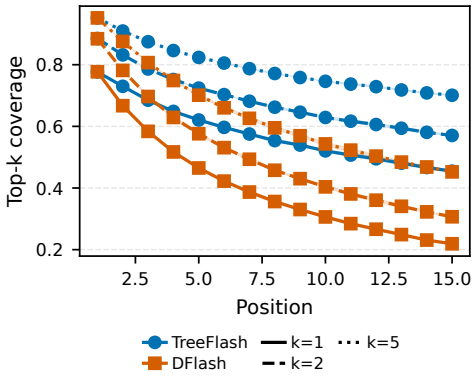


Figure 4: Coverage of target probability space with drafter top-K tokens for Qwen3 4B. As expected, both DFlash and TreeFlash have the same high coverage of 0.88 at the first token with just a budget of 2. However, the further the token is in the future, the more pronounced the benefit of AR-approximation becomes. Already for the third token, TreeFlash's top-2 coverage is 0.79 compared to DFlash's 0.69. At depth 15, TreeFlash's top-1 coverage is 0.45, which is slightly larger than DFlash's top-5 coverage of 0.44.

et al., 2021). Code generation is assessed on *HumanEval* (Chen et al., 2021) and *MBPP* (Austin et al., 2021). General instruction-following ability is evaluated using *MT-Bench* (Zheng et al., 2023). Each dataset is limited to 64 samples, with the exception of MT-Bench, which is evaluated on 80 tasks.

Training Setup. Taking inspiration from DFlash (Chen et al., 2026), we train on a 100k-sample subset of a synthetic dataset generated from the Nemotron Post-Training Dataset V2 (Nathawani et al., 2025) and CodeAlpaca (Chaudhary, 2023) prompts. All models are trained for one epoch with an effective batch size of 128 and 128 anchors per sample. We use a linear learning rate warmup for 128 steps, followed by a cosine decay schedule. Optimization is performed using AdamW with a peak learning rate of 10^{-4} (Loshchilov and Hutter, 2019). Loss scaling uses a factor of 7 as in DFlash, and sequences are limited to 3072 tokens.

Metrics. In speculative decoding, the main metric of interest is *speedup*, which measures the increase in throughput compared to vanilla decoding. This metric, however, depends on multiple factors, such as implementation and hardware. Therefore, a commonly used metric for measuring drafter quality is *block efficiency* (τ), which measures the number of accepted tokens plus the additional residual token generated from the verification step. This is equivalent to the average number of tokens generated per draft-verify iteration. Furthermore, to measure drafter quality, we report the *total variation distance (TVD)* compared to the verifier's autoregressive distribution $p(x_t|x_{<t})$. Furthermore, we report *top-K coverage*, which measures the cumulative probability of the tokens in the drafter's top-K set under the verifier distribution. See Appendix A for details on the calculation of TVD and top-K coverage.

Inference Parameters. We use a block size of $\gamma = 16$, and for TreeFlash, we use $M = 16$ for the tree construction. Drafter temperatures are set to $T = 1$ regardless of the verifier temperature. We limit the maximum output length to 2048 additional tokens for all samples. We use PyTorch SDPA as the attention implementation during inference and BFloat16 precision. Experiments are conducted on NVIDIA GH200 GPUs.

4.1 Results

Speculative Decoding. Tables 1 and 2 report speedup and block efficiency across all targets, tasks, and decoding regimes. Figure 7 shows a qualitative example of a drafted tree comparing DDTree's marginally guided construction with TreeFlash's AR-approximation guided draft tree. TreeFlash consistently outperforms all baselines

Method	HumanEval Speedup	HumanEval τ	MBPP Speedup	MBPP τ
T=1 (Sampling)				
DFlash (16)	$4.69^{0.07}$	$6.41^{0.06}$	$4.98^{0.17}$	$6.65^{0.20}$
DDTree (16)	$5.25^{0.21}$	$7.23^{0.23}$	$5.73^{0.03}$	$7.66^{0.07}$
TreeFlash (16)	$\mathbf{5.74}^{0.26}$	$\mathbf{7.97}^{0.30}$	$\mathbf{6.03}^{0.08}$	$\mathbf{8.21}^{0.10}$
DDTree (64)	$6.31^{0.03}$	$8.61^{0.01}$	$6.59^{0.07}$	$8.84^{0.12}$
TreeFlash (64)	$\mathbf{6.98}^{0.15}$	$\mathbf{9.68}^{0.12}$	$\mathbf{7.20}^{0.07}$	$\mathbf{9.82}^{0.10}$
T=0 (Greedy)				
DFlash (16)	5.50	7.43	5.41	7.19
DDTree (16)	6.23	8.43	5.86	7.87
TreeFlash (16)	**6.56**	**8.99**	**6.20**	**8.50**
DDTree (64)	7.14	9.66	6.80	9.12
TreeFlash (64)	**7.69**	**10.57**	**7.47**	**10.22**

Table 2: Block efficiency (τ) and speedup for Qwen3 Coder 30B A3B. Notation follows Table 1. TreeFlash consistently outperforms both DFlash and DDTree across all budgets and temperature settings, confirming that the gains observed on the 4B and 8B target models transfer to a substantially larger MoE architecture. At the budget-matched setting (B=16), TreeFlash improves over DFlash by $+1.50$ in τ and $+0.99\times$ in speedup. Under the increased budget (B=64), $+42.8\%$ speedup over DFlash, with AR-approximation responsible for $+1.01$ block efficiency over DDTree.

$T=$	MATH-500 0	MATH-500 1	HumanEval 0	HumanEval 1	MT-Bench 0	MT-Bench 1	Mean 0	Mean 1
DFlash	7.9	$5.9^{0.1}$	6.4	$5.6^{0.0}$	3.5	$3.1^{0.0}$	5.9	$4.9^{0.0}$
w/o Tree	**8.3**	$\mathbf{6.2}^{0.1}$	**6.9**	$\mathbf{6.0}^{0.2}$	**3.7**	$\mathbf{3.2}^{0.0}$	**6.3**	$\mathbf{5.1}^{0.1}$
DDTree	9.8	$7.8^{0.1}$	8.5	$7.6^{0.2}$	4.9	$4.5^{0.0}$	7.7	$6.6^{0.1}$
w/o AR-app.	9.6	$7.8^{0.1}$	8.4	$7.7^{0.1}$	4.9	$4.4^{0.0}$	7.6	$6.6^{0.1}$
w/ Linear	10.1	$8.1^{0.1}$	8.6	$7.9^{0.1}$	5.0	$4.6^{0.0}$	7.9	$6.8^{0.1}$
w/ Frozen	11.0	$8.8^{0.1}$	9.3	$8.5^{0.1}$	5.4	$4.9^{0.0}$	8.6	$7.4^{0.0}$
w/ 2-prev	11.0	$8.8^{0.0}$	9.5	$8.6^{0.2}$	**5.5**	$4.9^{0.1}$	**8.7**	$7.4^{0.1}$
w/ CE	10.9	$8.8^{0.2}$	9.5	$8.6^{0.0}$	5.4	$4.9^{0.0}$	8.6	$7.4^{0.1}$
w/o Scaling	**11.0**	$\mathbf{8.9}^{0.1}$	**9.6**	$8.6^{0.1}$	5.3	$4.9^{0.0}$	8.6	$7.4^{0.0}$
TreeFlash	11.0	$8.7^{0.0}$	9.4	$\mathbf{8.7}^{0.0}$	5.4	$\mathbf{5.0}^{0.0}$	8.6	$\mathbf{7.5}^{0.0}$

Table 3: Block efficiency for the ablation experiments. DFlash and *w/o Tree* use sequence drafts; all other rows use tree drafts with $B = 64$. Results are on 64 samples for *MATH-500*, *HumanEval*, and *MT-Bench*. *w/o Tree* uses TreeFlash's AR-approximation while retaining sequence drafting, *w/o AR-app*roximation is an extended finetune using the DFlash paradigm, *w/ Linear* replaces the SwiGLU layer with a linear layer, *w/ Frozen* keeps the DFlash backbone frozen during training, *w/ 2-prev* inputs the two previous tokens to the AR-approximation, *w/ CE* replaces the KL loss with cross-entropy, and *w/o Scaling* removes the loss scaling scheme from DFlash training. Using a SwiGLU layer instead of a linear layer provides the largest improvement in block efficiency. Further, keeping the backbone frozen has similar performance to full TreeFlash, indicating that the majority of the gains are attributable to the AR-approximator itself rather than continued backbone fine-tuning.

in every evaluated configuration. As can be seen in Tables 1 and 2, under a budget-matched setting (B=16), TreeFlash improves over DFlash by an average of $+1.35$ in τ ($+24.8\%$) and $+0.71\times$ in speedup ($+17.8\%$).

The major driver of this improvement is the transition from chain to tree drafting. As can be seen in Tables 1 and 2, DDTree alone recovers $\sim$63% of the block-efficiency gain and $\sim$73% of the speedup gain that TreeFlash achieves over DFlash. Nevertheless, AR-approximation contributes a consistent and meaningful improvement on top: as can be seen in Tables 1 and 2, TreeFlash outperforms DDTree by $+0.50$ in τ ($+7.6\%$) and $+0.19\times$ in speedup ($+4.0\%$) under matched budget.

A key advantage of tree drafting is that it enables scaling B beyond the block size. As can be seen in Tables 1 and 2, under the increased budget of B=64, DDTree already substantially improves over DFlash by $+34.9\%$ τ and $+34.3\%$ speedup on average. TreeFlash pushes this further to $+51.6\%$ and $+46.4\%$ respectively.

Notably, the benefit of AR-approximation grows with draft budget B. As can be seen in Tables 1 and 2, at B=16 TreeFlash improves over DDTree by $+7.6\%$ in τ and $+4.0\%$ in speedup, whereas at B=64 these gains increase to $+12.4\%$ and $+9.1\%$ respectively. We attribute this to the fact that larger budgets expose more tokens at deeper draft positions. As shown in Figures 3 and 4, this is where AR-approximation provides the greatest improvement over marginal distributions. In line with this, the benefit of AR-approximation is most pronounced in high-acceptance regimes. As can be seen in Table 1, on MATH-500 and $B = 64$ TreeFlash improves τ over DDTree by $+13.3\%$, whereas on MT-Bench the gain is only $+10.4\%$.

In summary, TreeFlash consistently and meaningfully outperforms both DFlash and DDTree across all evaluated models, tasks, and decoding regimes. While the transition to tree-based drafting provides the largest single boost in performance, AR-approximation contributes a consistent, complementary gain that grows with draft budget and is most pronounced in high-acceptance settings.

Autoregressive Approximation. TVD to the target distribution serves as a useful diagnostic for the calibration between drafter and verifier.

As shown in Figure 3, the TVD of DFlash grows steadily with draft depth, starting at 0.19 for the

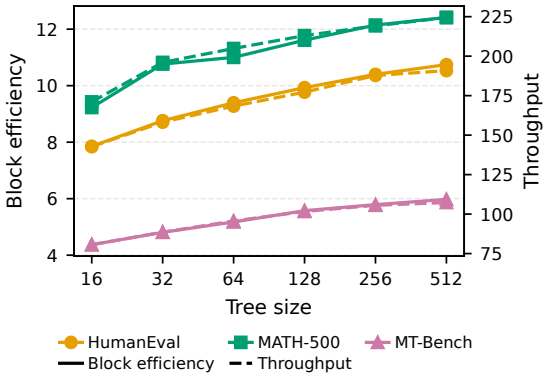


Figure 5: Block efficiency and throughput for different draft budgets B for TreeFlash and Qwen3 4B with $T{=}0$. Increasing the tree budget improves both metrics, with mean block efficiency growing from 7.15 at $B{=}16$ to 9.71 at 512. Note that in multi-batch settings the budget can't be scaled to large values, as verification becomes compute-bound.

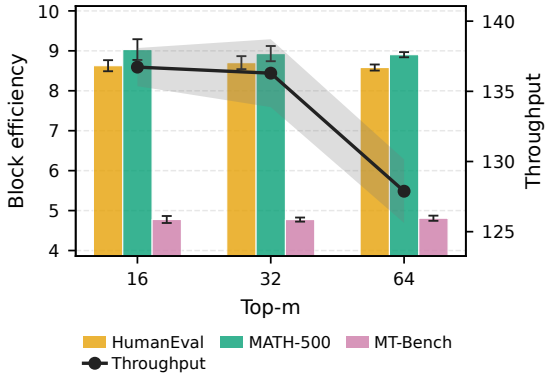


Figure 6: Block efficiency and throughput for different values of M for TreeFlash with Qwen3 4B as the target model under greedy decoding with $B{=}64$. In terms of block efficiency, the values are similar; however, for throughput, the cutoff appears to be around $M = 32$, where larger values incur too much overhead and cause throughput to drop. This value depends on the underlying hardware and implementation.

first drafted token and reaching 0.81 by the end of the block. Notably, this degradation is not specific to DFlash but reflects a systematic limitation of marginal distributions. As can be seen, the true marginal $p(x_{t+i} \mid x_{\leq t})$ (see Appendix A) also exhibits increasing divergence with depth. TreeFlash is not immune to this effect, but its growth in TVD is substantially slower, topping out at 0.62. Remarkably, beyond the 9th draft position, the true marginal itself exhibits higher TVD than TreeFlash, suggesting that conditioning on the preceding token yields a better calibrated distribution compared to the ground-truth marginal.

Figure 4 shows top-K coverage, which correlates the number of alternative tokens in the draft tree to the acceptance probability. Again, for tokens at depth 1, TreeFlash and DFlash both achieve a top-1 coverage of 0.78. As depth increases, however, the gap widens considerably: at depth 10 and beyond, TreeFlash's top-1 coverage approximately matches DFlash's top-5 coverage. Concretely, this means that a single TreeFlash candidate achieves the same acceptance probability as 5 DFlash candidates, allowing for more efficient token utilization at depth.

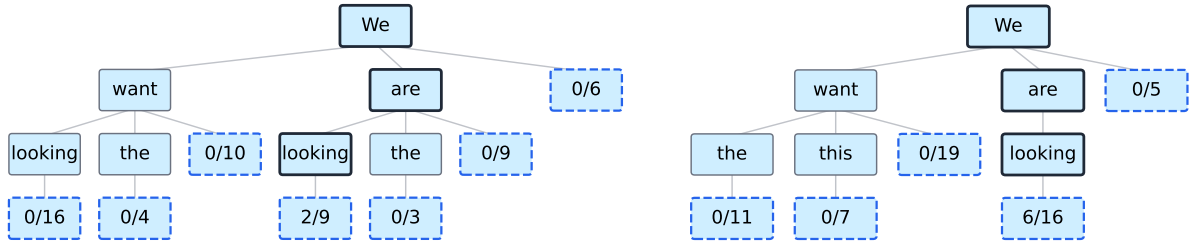
In conclusion, the AR-approximation's benefits can be observed not only in empirical speculative decoding measures but also in distributional metrics. We show that TreeFlash's distribution is not only better than DFlash's, but also better than the true marginal distribution, suggesting that the gained information from conditioning on the preceding token captures token-level coherence that marginal distributions systematically fail to exploit.

4.2 Ablations

Ablation Setup. All ablation experiments are conducted on Qwen3 4B. The training setup is equivalent to the main experiments, unless stated otherwise. Evaluation is performed on 64 samples of MATH-500, HumanEval, and MT-Bench. Tree-based ablations use $B{=}64$, while the sequence-drafting comparison uses $B{=}16$.

Model Design. The design of the AR-approximator is critical to the efficacy of TreeFlash: it must be both lightweight enough to preserve the $\mathcal{O}(1)$ drafting complexity and expressive enough to meaningfully correct the hidden states. First, we finetune the DFlash model with no AR-approximation mechanism, which, as can be seen in Table 3, fails to improve over the initial checkpoint. As an alternative to the SwiGLU head, we evaluate a simple linear map that applies a bias to the hidden state conditioned on the previous token. As shown in Table 3, this linear approximator consistently underperforms TreeFlash in block efficiency across all evaluated settings, confirming that the non-linear capacity of the SwiGLU head is necessary.

We further show that a single previous token is sufficient for AR-approximation. As shown in Table 3, conditioning on the previous bigram (w/ 2-prev) provides little to no improvement over



(i) DDTree *(ii) TreeFlash*

Figure 7: Qualitative comparison of decoding sub-trees produced by DDTree and TreeFlash. Outline tokens indicate acceptance, and i/j denotes i nodes accepted from j total. Note that TreeFlash produces more coherent bigrams, allowing for better utilization of draft budget. See Appendix C for more details.

TreeFlash. Given the increase in complexity and growth of evaluations required, we conclude that single-token conditioning strikes the best balance between performance and efficiency.

Employing AR-approximation with sequence drafting (w/o Tree) also provides improvement over DFlash's marginal distribution.

Training Paradigm. TreeFlash jointly fine-tunes both the AR-approximator and the DFlash backbone. To isolate the contribution of the AR-approximator, we evaluate a frozen-backbone variant (*w/ Frozen*), which keeps the DFlash checkpoint fixed and trains only the SwiGLU head. As shown in Table 3, this variant performs only marginally worse than full TreeFlash, indicating that the majority of the gains are attributable to the AR-approximator itself rather than continued backbone fine-tuning.

Diverging from DFlash, TreeFlash uses forward KL divergence rather than cross-entropy to align the drafter to the target distribution. Table 3 shows that replacing the KL divergence with cross-entropy (*w/ CE*) yields comparable performance, with TreeFlash retaining a slight edge. Like DFlash, TreeFlash applies a loss scaling mechanism that penalizes early errors more heavily than later errors. Removing this scaling (*w/o Scaling*) produces mixed results: block efficiency is marginally higher without scaling on long-acceptance tasks such as MATH-500 and code, while scaling provides a small benefit in low-acceptance regimes such as MT-Bench.

Overall, the training design choices each contribute modest improvements, showing that their impact is secondary to the architectural choices analyzed above.

Inference Parameters. The two primary inference-time hyperparameters of TreeFlash are the drafter token budget B and the candidate set size M.

The token budget B controls the node count of the drafted tree, and larger values are expected to increase block efficiency τ at the cost of additional verification compute. As shown in Figure 5, B can be scaled to large values without the verification overhead outweighing the gains in block efficiency. In practice, the optimal B depends on multiple factors, such as batch size, hardware, and memory constraints, and should be tuned accordingly.

The hyperparameter M controls the number of candidate tokens considered per position during tree construction, and directly determines the number of evaluations required by the AR-approximator. As shown in Figure 6, block efficiency is comparable across different M values, while throughput favors $M \leq 32$. Therefore, we recommend choosing $M \leq 32$, as these values provide the best throughput in our experiments.

5 Conclusion

This paper presents TreeFlash, a method to improve one-shot block-wise drafting by incorporating a lightweight AR-approximation mechanism that conditions each draft position on the preceding token, preserving the $\mathcal{O}(1)$ decoding complexity of parallel drafters while addressing the distributional gap that grows with draft depth. TreeFlash achieves consistent and substantial performance improvements over both DFlash and DDTree across a wide range of tasks, model sizes, and decoding regimes, improving block efficiency by $+12.4\%$ over trees constructed with fully marginal distributions.

Limitations

TreeFlash conducts all evaluations in a single-batch setting using SDPA attention. Production serving systems, however, typically incorporate a range of optimizations, such as quantization, multi-batch decoding, and custom attention kernels, that can conflict with large tree sizes and dynamic tree topologies (Dao et al., 2022; Lin et al., 2024; Kwon et al., 2023). The interaction between TreeFlash and such serving-level optimizations is not evaluated in this work.

TreeFlash is initialised from an existing DFlash checkpoint rather than trained from scratch. A full pretraining run is beyond the scope of this paper, and would likely require an auxiliary loss on the initial candidate distribution used to construct the M-ary tree. The same concern applies to extended training of the AR-approximator: prolonged finetuning of the DFlash backbone may degrade the initial candidate distributions used to construct the M-ary tree, leading to suboptimal final tree construction.

The AR-approximator is trained with teacher-forced input embeddings, which is the standard paradigm for autoregressive models but may be suboptimal here: at inference time, the approximator conditions on its own previously drafted tokens rather than ground-truth ones. Addressing this exposure bias is left to future work. Recent work on training models explicitly for tree generation rather than sequential AR modelling may also offer a promising future direction in training TreeFlash (Hu et al., 2026).

TreeFlash is evaluated exclusively on Qwen-family models. Whether the efficiency gains transfer to other model families remains an open question.

References

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. 2024. Hydra: Sequentially-dependent draft heads for Medusa decoding. *arXiv preprint arXiv:2402.05109*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

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

Sahil Chaudhary. 2023. Code alpaca: An instruction-following LLaMA model for code generation. https://github.com/sahil280114/codealpaca.

Jian Chen, Yesheng Liang, and Zhijian Liu. 2026. DFlash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

Tencent AngelSlim Project Contributors. 2025. AngelSlim.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*.

Shijing Hu, Jingyang Li, Zhihui Lu, and Pan Zhou. 2026. Bridging draft policy misalignment: Group tree optimization for speculative decoding. In *International Conference on Learning Representations*. Poster.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR.

Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. 2025a. DiffuSpec: Unlocking diffusion language models for speculative decoding. *arXiv preprint arXiv:2510.02358*.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025b. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*. Poster.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*.

Fuliang Liu, Xue Li, Ketai Zhao, Yinxi Gao, Ziyan Zhou, Zhonghui Zhang, Zhibin Wang, Wanchun Dou, Sheng Zhong, and Chen Tian. 2026. DART: Diffusion-inspired speculative decoding for fast LLM inference. *arXiv preprint arXiv:2601.19278*.

Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Jan Kautz, Ce Zhang, and Pavlo Molchanov. 2025. TiDAR: Think in diffusion, talk in autoregression. *arXiv preprint arXiv:2511.08923*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

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. 2024. 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*. Association for Computing Machinery.

Dhruv Nathawani, Shuoyang Ding, Vitaly Lavrukhin, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Boris Ginsburg, and Jane Polak Scowcroft. 2025. Nemotron-Post-Training-Dataset-v2.

Liran Ringel and Yaniv Romano. 2026. Accelerating speculative decoding with block diffusion draft trees. *arXiv preprint arXiv:2604.12989*. Concurrent work.

Jameson Sandler, Jacob K. Christopher, Thomas Hartvigsen, and Ferdinando Fioretto. 2025. SpecDiff-2: Scaling diffusion drafter alignment for faster speculative decoding. *arXiv preprint arXiv:2511.00606*.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, and 1 others. 2025. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. 2023. SpecTr: Fast speculative decoding via optimal transport. In *Advances in Neural Information Processing Systems*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. 2025. OPT-tree: Speculative decoding with adaptive draft tree structure. *Transactions of the Association for Computational Linguistics*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2024. Distillspec: Improving speculative decoding via knowledge distillation. In *International Conference on Learning Representations*, volume 2024, pages 32011–32050.

A TVD and Coverage approximation

To compute the target marginal distribution $p(x_{t+i} \mid x_{\leq t})$, we employ Monte Carlo estimation. For a given anchor position t, we draw N independent continuations $x^{(j)}_{t+1:t+\gamma}$, $j \in [N]$, and approximate the marginal as

$$\tilde{p}(x_{t+i} \mid x_{\leq t}) \approx \frac{1}{N} \sum_{j=1}^{N} p\Big(x_{t+i} \mid x^{(j)}_{<t+i}\Big). \quad (2)$$

, where $x^{(j)}_{<t+i-1} := x_{\leq t}, x^{(j)}_{t+1:t+i-1}$ Note that this is an unbiased estimator of the true marginal, i.e., $\mathbb{E}\left[\tilde{p}(x_{t+i} \mid x_{\leq t})\right] = p(x_{t+i} \mid x_{\leq t})$. The TVD between the target marginal and the target distribution is approximated with:

$$\frac{1}{N} \sum_{j=1}^{N} TVD\left(\tilde{p}(x_{t+i} \mid x_{\leq t}),\ p\Big(x_{t+i} \mid x^{(j)}_{<t+i}\Big)\right) \quad (3)$$

Note that while not an unbiased estimator, this serves as a lower bound to the ground truth TVD by Jensen's Inequality.

Top-K coverage is defined as follows: let $\hat{x}^{(1)}_{t+i}, \ldots, \hat{x}^{(k)}_{t+i}$ denote the top-K tokens according to the drafter; coverage at depth i is then defined as

$$C_k = \sum_{l=1}^{k} p\Big(x_{t+i} = \hat{x}^{(l)}_{t+i} \mid x_{<t+i}\Big). \quad (4)$$

For all evaluations, we sample 512 examples from the held-out validation set, drawing 16 anchor positions per sample with N=64 continuations each. To limit memory consumption, autoregressive probabilities are truncated to the top-256 tokens. All results are averaged over three random seeds, with standard deviations reported in the figures.

B Tree Construction

Figure 8 is the algorithm used for efficient tree construction. Note that Part 1 can be done efficiently in parallel on the GPU and Part 2 is done iteratively on the CPU.

1. Compute AR-approximated token distributions.
$(h_{t+1}, \ldots, h_{t+\gamma}) \leftarrow \mathrm{DFlash}(x_{\leq t})$
for $i \in [\gamma]$ **do**
  $q_{t+i} \leftarrow \mathrm{LMHead}(h_{t+i})$
  $\hat{x}^{(1:M)}_{t+i} \leftarrow \mathrm{TopM}(q_{t+i})$
  $e^{(m)}_{t+i} \leftarrow \mathrm{Embd}(\hat{x}^{(m)}_{t+i}), \quad m \in [M]$
end for
$h'_{t+1} \leftarrow h_{t+1} + \mathrm{SwiGLU}(\tilde{h}_{t+1} :: e_t)$
$q'_{t+1} \leftarrow \mathrm{LMHead}(h'_{t+1})$
for $i \in \{2, \ldots, \gamma\}$ **do**
  for $m \in [M]$ **do**
    $h'^{(m)}_{t+i} \leftarrow h_{t+i} + \mathrm{SwiGLU}(\tilde{h}_{t+i} :: e^{(m)}_{t+i-1})$
    $q'^{(m)}_{t+i} \leftarrow \mathrm{LMHead}(h'^{(m)}_{t+i})$
  end for
end for

2. Construct the draft tree with OPT-Tree selection.
$\mathcal{T} \leftarrow \{x_t\}$
$Q \leftarrow \mathrm{InitQueue}(q'_{t+1})$ – candidates keyed by path probability
while $|\mathcal{T}| < B + 1$ **do**
  $v_{t+i} \leftarrow \mathrm{PopMax}(Q)$
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{v_{t+i}\}$
  if $v_{t+i} = \hat{x}^{(m)}_{t+i}$ for some $m \in [M]$ **then**
    $Q \leftarrow Q \cup q'^{(m)}_{t+i+1}$
  end if
end while
return $\mathcal{T}$

Figure 8: AR-approximated OPT-Tree Construction. γ is the maximum draft depth, M is the hyperparameter controlling the size of the initial tree, and B is the tree/draft budget.

C Qualitative Trees

Tree Overlap. To quantify how AR-approximation changes the drafted tree, we compare DDTree and TreeFlash trees across multiple draft budgets. As shown in Table 4, the overlap between the two trees decreases as the draft budget increases from 68.0% shared non-root nodes at B=16 to 31.8% at B=512.

Qualitative Comparison. As illustrated in Figures 9 and 10, the two methods produce qualitatively different tree topologies. Because DDTree relies solely on the marginal distribution, the token ranking at each depth is identical across all paths. This forces DDTree to always produce a *nested* tree: for any two sibling nodes i and j at depth d, if i has a higher marginal probability than j, then the set of descendants of j must be a subtree of the descendants of i. Put differently, siblings at depth d are constrained to share the same child ordering, since no path-specific information is available to differentiate them. As shown in Figure 10, TreeFlash escapes this constraint: By conditioning on the draft context via the AR-approximator, different paths can yield different child rankings, allowing the tree to better reflect the true conditional structure of the continuations and resulting in better-calibrated topologies.

D Licenses

All Qwen3 model checkpoints, MT-Bench, and CodeAlpaca are released under the Apache 2.0 license. MATH-500, GSM8K, HumanEval, and the DFlash checkpoint are released under the MIT li-

B	Shared nodes
16	68.0%
32	56.5%
64	48.0%
128	41.5%
256	36.2%
512	31.8%

Table 4: Fraction of non-root nodes shared by DDTree and TreeFlash candidate trees on Qwen3 4B.

cense. MBPP is released under CC-BY-4.0; the Nemotron Post-Training Dataset V2 is predominantly released under CC-BY-4.0, with small subsets under ODC-BY and CC-BY-SA.

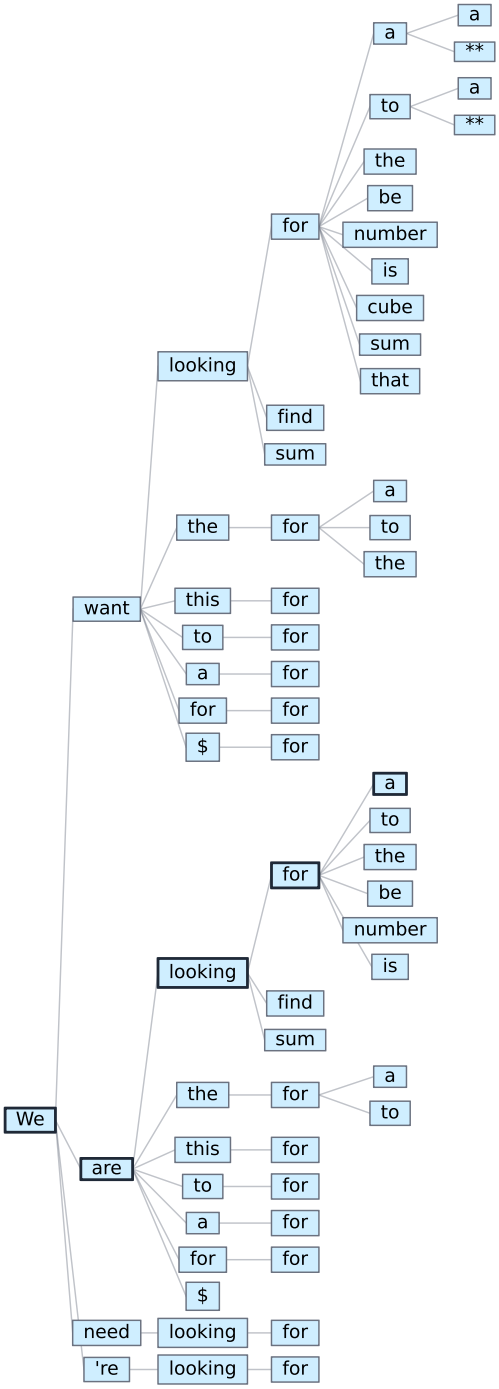


Figure 9: Example of a Draft Tree with $B{=}64$ produced by DDTree. Outlined tokens are accepted by Qwen3 8B under greedy sampling.

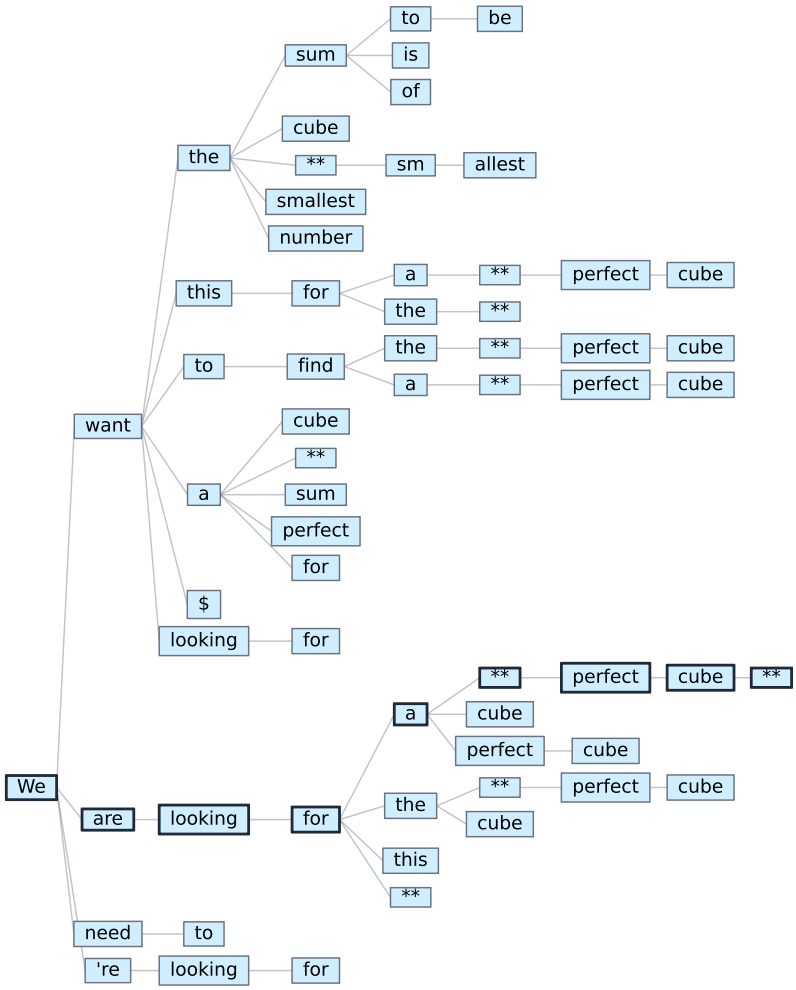


Figure 10: Example of a Draft Tree with $B{=}64$ produced by TreeFlash. Outlined tokens are accepted by Qwen3 8B under greedy sampling.