JETSPEC: BREAKING THE SCALING CEILING OF SPECULATIVE DECODING WITH PARALLEL TREE DRAFTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding (SD) accelerates autoregressive Large Language Models (LLMs) by drafting multiple tokens and verifying them in parallel, but it faces a scaling limitation: increasing the draft budget improves speed only when acceptance remains high and drafting overhead stays low. This ceiling has been difficult to break because prior head-based SD methods face a causality-efficiency dilemma. Autoregressive drafters produce path-conditioned candidates that are effective for tree speculative decoding with higher acceptance length, but their drafting cost grows with tree depth. Bidirectional block-diffusion drafters generate all positions in one pass, but their branch-agnostic marginals can form individually plausible yet mutually inconsistent trees, wasting budget and reducing acceptance. We propose JETSPEC, a head-based SD framework that combines one-forward drafting efficiency with causal information flow across draft representations. JetSpec trains a causal parallel draft head using fused hidden states from the frozen target model, producing causally coupled draft distributions whose scores better align with the target model's autoregressive factorization. This enables JetSpec to convert larger draft budgets into longer accepted prefixes and higher end-to-end speedup. Across math, coding, and chat benchmarks on dense and MoE Qwen3 models, JetSpec consistently outperforms bidirectional-head and tree-based SD baselines. On H100 GPUs, JetSpec achieves up to $9.64\times$ speedup on MATH-500 and $4.58\times$ on open-ended conversational workloads, with further latency gains demonstrated through vLLM integration under realistic serving loads.

1 INTRODUCTION

Modern autoregressive (AR) Large Language Models (LLMs) operate under a simple yet costly bottleneck: decoding remains largely serial. This sequential generation process makes latency a major challenge when they are deployed in applications such as math (Wei et al., 2022; Zhu et al., 2025), coding (Deng et al., 2025; Jain et al., 2024) and other agentic reasoning tasks (Patil et al., 2025; Yao et al., 2025; Lu et al., 2025), where models often perform long generations.

Existing approaches address this challenge by either incorporating multi-token prediction (MTP) during pre-training or mid-training (Gloeckle et al., 2024; DeepSeek-AI, 2024; 2026; AI, 2026; Xiaomi, 2026), or post-training methods that adapt pretrained LLMs into diffusion LLMs (dLLMs) at the costs of varying degrees of performance degradation (Zhao et al., 2025; Wu et al., 2025). In contrast, applying speculative decoding (SD) at inference time to reduce generation latency while preserving quality (Leviathan et al., 2023; Chen et al., 2023).

Among these techniques, SD stands out for its low adaptation cost and simplicity. On one hand, prior work shows that SD performance can be substantially improved through lightweight draft-model alignment, oftentimes less than 1B training tokens (Liu et al., 2024; Zhou et al., 2024; Goel et al., 2024). On the other hand, head-based methods such as Medusa and EAGLE reduce deployment complexity by avoiding the conventional separate draft-target model setup (Cai et al., 2024; Li et al., 2024a). More recent head-based methods such as EAGLE-3 (Li et al., 2025) and DFlash (Chen et al., 2026) continue to improve drafting quality and reducing drafting cost for better end-to-end acceleration. Despite these advances, head-based SD still faces a causality-efficiency dilemma. Autoregressive drafters such as EAGLE produce high-quality path-conditioned candidates but require sequential draft passes as tree depth grows. Bidirectional block-diffusion drafters such as DFlash

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

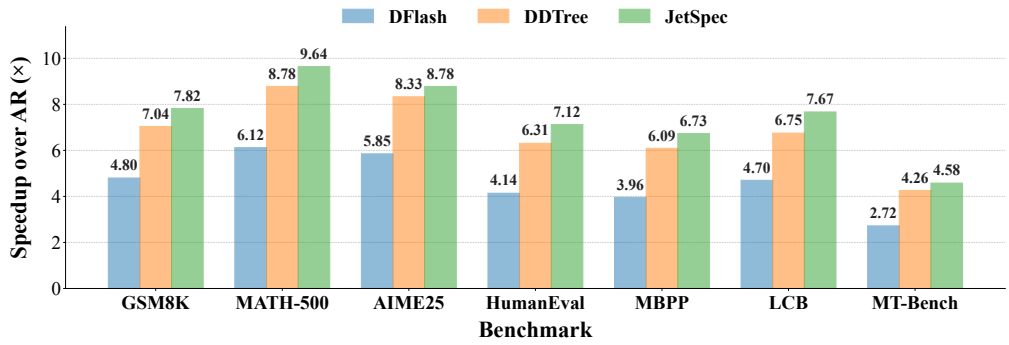


Figure 1: End-to-end decoding speedup over standard autoregressive decoding on H100 GPUs across math, coding, and chat benchmarks. DFlash denotes the original block-parallel drafting method, DDTree is tree-based variant of DFlash, and JetSpec denotes our method. Both employ a tree budget of 256 tokens using Algorithm 1.

produces drafts in one pass, but their branch-agnostic marginals can form individually plausible yet mutually inconsistent trees, wasting budget and reducing acceptance.

We propose JETSPEC, a SD framework designed to break this causality–efficiency dilemma: JetSpec trains a causal parallel draft head to predict distributions for multiple draft depths in one forward pass while enforcing left-to-right information flow across draft-position hidden states. This produces causal depth-wise marginals that better reflect autoregressive continuation structure, enabling tree acceptance to scale more effectively with draft budget.

Empirically, JetSpec demonstrates significant efficiency gains. In the low-token-budget regime with 16 draft tokens, JetSpec achieves competitive end-to-end speedup. In the high-token-budget regime with 256 draft tokens, JetSpec achieves strong gains, reaching $\tau = 10.7$ and $9.5\times$ speedup on MATH-500, and remaining effective on MT-Bench with $\tau = 5.9$ and over $4\times$ speedup. We further integrate JetSpec into industry-grade serving engine and evaluate it across batch sizes, where it consistently outperforms baselines under small to moderate serving loads, with especially strong gains on higher-end GPUs such as H100 and B200. These results highlight the practical potential of JetSpec for reducing decoding latency in serving.

In a nutshell, our work makes the following contributions:

- We propose JETSPEC, a paradigm designed to jointly optimize drafting cost and acceptance rate for speculative decoding. JetSpec trains a causal prediction head with parallel tree-drafting capability, enabling low-cost draft generation while preserving causal dependencies across draft-depth representations.
- We develop and evaluate tree-drafting algorithms that allow speculative decoding to better leverage additional decoding compute. By scaling the draft token budget, JetSpec achieves up to $\tau = 10.7$ and more than $9.5\times$ end-to-end speedup.
- We integrate JetSpec into an industry-grade serving engine and evaluate it under realistic serving scenarios. JetSpec consistently outperforms baselines under small to moderate serving loads, highlighting its practical potential for reducing latency in real-world deployments.

2 RELATED WORK

Speculative Decoding. Speculative decoding accelerates autoregressive generation by drafting tokens and verifying them with the target model (Leviathan et al., 2023; Chen et al., 2023). Its speedup depends on draft quality, which controls accepted length, and drafting cost, which controls wall-clock gain. Prior work improves these factors via draft-target alignment (Zhou et al., 2024; Liu et al., 2024; Goel et al., 2024), head-based drafters such as Medusa and EAGLE (Cai et al., 2024; Li et al., 2024a; 2025), or draft-free retrieval methods such as Prompt Lookup Decoding (Saxena, 2023), REST (He et al., 2024), PLD+ (Somasundaram et al., 2024), SAM Decoding (Hu et al., 2025b),

and SuffixDecoding (Oliaro et al., 2024). While these methods reduce overhead, they often rely on semantic overlaps for retrieval, or repeated patterns. In contrast, JetSpec targets low-cost parallel drafting with a learned causal head that preserves branch-wise dependencies for high-acceptance tree verification.

Drafter Alignment. Previous works train draft models or draft heads to better match the target model. Multi-token prediction objectives supervise future-token predictions during pretraining or midtraining (Gloeckle et al., 2024; DeepSeek-AI, 2024), while post-training approaches use supervised fine-tuning, offline distillation, online distillation, or direct acceptance-rate optimization to improve agreement between draft and target distributions (Zhou et al., 2024; Liu et al., 2024; Goel et al., 2024). These methods primarily improve the alignment term that controls acceptance rate. JetSpec is complementary: it trains the drafter not only to match the target model, but also to preserve directional structure for budgeted tree construction, so tree expansion uses causally coupled draft distributions rather than independent position marginals.

Parallel Decoding and Drafting. Parallel decoding methods reduce latency by relaxing strict one-token-at-a-time generation. Self-speculative decoding drafts with early target-model layers and verifies with later layers (Elhoushi et al., 2024). Diffusion LLMs refine multiple tokens per iteration (Wu et al., 2025; Qian et al., 2026). Jacobi-style methods preserve causality by parallel fixed-point updates, with recent consistency distillation works improving convergence to autoregressive outputs (Fu et al., 2024; Kou et al., 2024; Hu et al., 2025a). These methods increase tokens committed per forward, but are typically applied to the target model itself. DFlash is the closest baseline, generating multiple draft tokens in one block-parallel pass (Chen et al., 2026); however, its branch-agnostic block-diffusion predictions are not explicitly optimized for causal tree construction.

3 JETSPEC

3.1 BACKGROUND

Scaling Bottlenecks in Speculative Decoding. In speculative decoding (SD), a lightweight draft model M_q proposes N tokens, which are verified in parallel by the target model M_p (Leviathan et al., 2023). The target accepts the longest prefix consistent with its predictions and resumes from the first rejected token. Let α denote the average acceptance rate and c the per-step cost of M_q relative to M_p. Under the standard i.i.d. acceptance assumption,

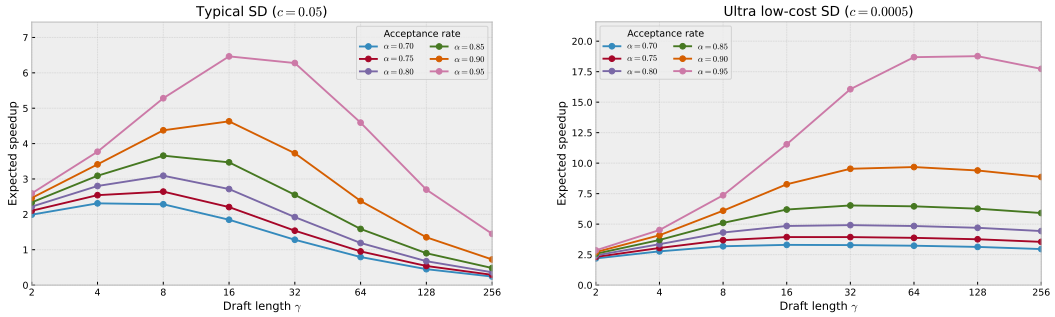
$$\mathbb{E}[\#\text{tokens}] = \frac{1-\alpha^{N+1}}{1-\alpha}, \qquad \text{Speedup} = \frac{1-\alpha^{N+1}}{(1-\alpha)(Nc+1)}. \tag{1}$$

Thus, scaling to longer drafts requires both high acceptance α and low total drafting cost Nc: as N grows, even modest acceptance degradation or drafting overhead can offset the gains from parallel verification. Figure 2 illustrates this tradeoff.

Existing work usually improves acceptance rate and drafting cost *separately*. To increase acceptance quality α, draft-model alignment methods improve agreement between M_q and M_p (Zhou et al., 2024; Liu et al., 2024; Goel et al., 2024), while tree SD methods improve verification efficiency by expanding multiple candidate branches in parallel (Miao et al., 2024; Li et al., 2024b). To reduce drafting cost c, head-based methods avoid a separately deployed draft model (Cai et al., 2024; Li et al., 2024a; 2025); more recently, DFlash further reduces overhead through block-diffusion-style drafting, generating multiple draft tokens in a single draft pass (Chen et al., 2026) (see Appendix G). However, scaling to longer drafts requires simultaneously maintaining high α and low c, motivating JetSpec to jointly improve acceptance rate and drafting efficiency.

3.2 CAUSAL PARALLEL DRAFTING

Can We Realize Both Parallel Drafting and Tree SD? A natural direction to co-optimize both aspects is to combine tree drafting for higher acceptance with parallel drafting for lower cost. However, existing parallel drafting methods such as DFlash can break the causal structure needed for effective tree construction: future-position marginals are produced without causal dependence on earlier draft positions, so high-probability marginals can compose into mutually inconsistent



(a) Typical SD ($c = 0.05$). (b) Ultra low-cost SD ($c = 0.0005$).

Figure 2: Expected speculative decoding speedup scales as a function of draft length γ, under different per-token drafting costs c and acceptance rates α. Comparing the two panels shows that reducing c substantially improves the scalability of speculative decoding with respect to γ, and increasing α further amplifies this effect. The results highlight that pushing per-token drafting cost c low and acceptance rate α high is critical for unlocking stronger scaling with longer draft lengths.

branches. JetSpec addresses this limitation with a causal parallel draft head that couples draft-depth representations left-to-right while preserving the efficiency of one-pass parallel prediction.

Why Causality Matters. A tree inherently encodes a causal structure. Consider a draft tree rooted at the original prefix x, where each node v corresponds to a candidate token y_v. Let $\pi(v) = (y_{v_1}, \ldots, y_v)$ denote the path from the root to v, and $\pi_{<v}$ its ancestor tokens. Ideally, the distribution at each node should reflect its corresponding branch prefix. In contrast, a bidirectional parallel drafter allows each position to attend to representations from both earlier and later draft positions. It creates a mismatch between bidirectional draft representations and the causal structure required for tree construction.

Architecture. JetSpec adopts a head-based drafting architecture, where a draft head can reuse intermediate features from the target model. Specifically, DFlash shows that a lightweight block-parallel draft head can reuse target-model hidden states and inject fused target-context features into the draft layers' KV cache for strong target-model guidance (Chen et al., 2026). Building on this design, JetSpec introduces a causal parallel decoding head for tree drafting.

Given a prefix x with fused target-model features h_x^o, the draft head predicts logits for multiple future positions in parallel. Following standard speculative decoding notation, the target model defines an autoregressive distribution over a length-k continuation:

$$p(y_{1:k} \mid x) = \prod_{i=1}^{k} p(y_i \mid x, y_{<i}). \tag{2}$$

To introduce analogous directional dependencies into parallel drafting, we apply a causal attention mask across draft positions. Position i may attend to the prefix features and earlier draft-position hidden states, but not to later positions. The mask and the masked attention at position i are

$$B_{i,j} = \begin{cases} 0, & j \le i, \\ -\infty, & j > i, \end{cases} \qquad \mathrm{Attn}(Q_i, K, V) = \mathrm{softmax}\left(\frac{Q_i K^\top}{\sqrt{d}} + B_i\right) V, \tag{3}$$

with prefix features visible to all draft positions. This allows all draft positions to be processed in parallel while enforcing directional information flow from earlier to later positions. Consequently, the distribution $r_i(\cdot \mid x, h_x^o)$ at position i is computed from a hidden representation that depends causally on earlier draft-position representations, rather than being produced independently across positions.

The causal mask induces directional dependencies across draft-position representations. Let h_i^q denote the draft-head hidden state at position i. Then

$$h_i^q = M_q\big(x, h_x^o, h_{<i}^q\big), \qquad q_i = \mathrm{softmax}(W h_i^q), \tag{4}$$

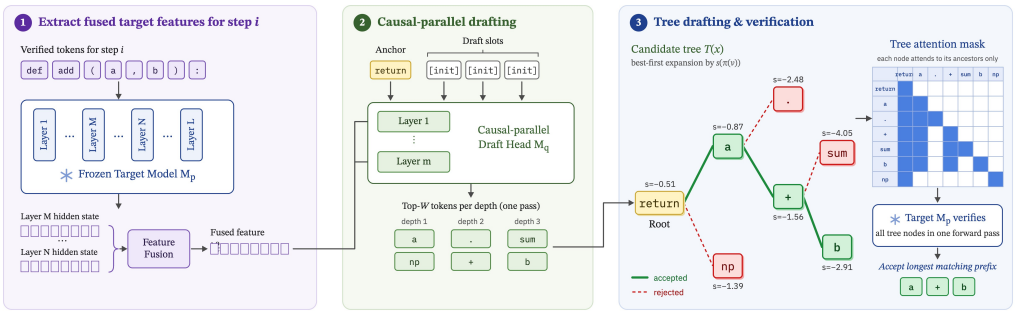


Figure 3: JetSpec design overview. JetSpec extracts fused hidden features from the frozen target model and conditions a causal-parallel draft head to generate high-quality candidate trees in one forward pass.

where $h^q_{<i}$ denotes earlier draft-position representations and q_i the token distribution at position i. Although all positions are predicted in parallel, each position causally incorporates earlier representations, introducing directional dependencies for tree drafting.

3.3 Training

Data Preparation. We train the draft head on target-aligned sequences, obtained either from the training corpus, as in multi-token prediction (Gloeckle et al., 2024; DeepSeek-AI, 2024), or from continuations regenerated by the target model (Agarwal et al., 2024; Gu et al., 2024; Liang et al., 2026). Given a sequence $(x, y_{1:k})$, we sample anchor positions and, at each anchor, construct a length-N training block of N consecutive future positions. The draft head predicts these N positions in parallel under a block-causal mask, matching the per-position teacher logits produced by the target model on the same ground-truth prefix.

Distillation Loss. We train the draft head with a soft-label distillation objective so that the draft distribution preserves the target model's relative preferences across multiple plausible continuations. Among distillation objectives, forward KL encourages the draft distribution to cover the target model's probability mass, preserving soft-label information beyond the top prediction. Reverse KL, in contrast, is more mode-seeking and can concentrate probability on high-confidence teacher modes (Gu et al., 2024; Agarwal et al., 2024). We find this matters in practice (Section 4): reverse KL underperforms substantially, while forward KL slightly outperforms hard-label SFT. We therefore adopt forward KL by default, and present the loss in this form below.

For each active draft position m, let $z_q^{(m)}$ and $z_p^{(m)}$ denote the draft and target logits over the vocabulary $\mathcal{V}$. We define temperature-normalized distributions $\tilde{q}^{(m)} = \mathrm{softmax}(z_q^{(m)}/T_{\mathrm{KD}})$ and $\tilde{p}^{(m)} = \mathrm{softmax}(z_p^{(m)}/T_{\mathrm{KD}})$, and minimize the forward KL

$$\mathcal{L}_{\mathrm{FKL}}^{(m)} = D_{\mathrm{KL}}\left(\tilde{p}^{(m)} \,\middle\|\, \tilde{q}^{(m)}\right). \tag{5}$$

The final objective is computed over all active draft-token positions induced from the training sequence and normalized by the active-position mask (see Appendix D for details):

$$\mathcal{L}_{\mathrm{train}} = T_{\mathrm{KD}}^2 \frac{\sum_m w_m \mathcal{L}_{\mathrm{FKL}}^{(m)}}{\sum_m w_m}, \tag{6}$$

where m indexes draft-token positions within sampled blocks, excluding non-loss and anchor-token positions, and $T_{\mathrm{KD}} > 0$ is the distillation temperature. This objective preserves the target model's soft-label preferences and provides teacher-aligned candidate distributions for tree expansion.

3.4 Drafting and Verification

Parallel Tree Drafting. An overview of the design is shown in Figure 3. At every decoding step i, the causal draft head conditions on the hidden states h_x^o of tokens that were verified and accepted

in the previous step, and constructs a candidate tree with maximum draft depth N, branching width W, and total node budget B. With one draft-head forward pass, we obtain logits for all draft depths and extract the top-W candidate tokens at each depth. Tree expansion is then driven by a branch scoring function $s(\pi(v))$, whose form can vary across implementations; we ablate different choices in Section 4.4. By default, we use accumulated draft log-probability for scoring:

$$s(\pi(v)) = \sum_{u \in \pi(v)} \log r_{d(u)} \left(y_u, \mid x, h_x^o\right), \tag{7}$$

where where $d(u)$ is node depth and r_d is the distribution produced at depth d by the one-pass causal head. We then repeatedly pop the highest-scoring expandable node, expand it with up to W children, and reinsert the children with updated scores until the budget is filled or no expandable node remains. This yields a candidate tree $\mathcal{T}(x) = \{\pi(v) \mid v \in \mathcal{V}_\mathcal{T}\}$, where each path $\pi(v)$ is a speculative continuation; details are given in Algorithm 1.

Tree Verification. After constructing the candidate tree $\mathcal{T}(x)$, the target model M_p verifies all tree nodes in parallel using tree attention. For any candidate branch $\pi(v) = y_{1:k}$, the target distribution factorizes autoregressively in compliance with Equation 2.

The verifier then applies the speculative decoding acceptance rule along each candidate branch. For each draft token y_t, we write the acceptance decision as

$$A_t \sim \text{Bernoulli}(\alpha_t), \qquad \alpha_t = \alpha(y_t;\, q(\cdot \mid x, y_{<t}), p(\cdot \mid x, y_{<t})), \tag{8}$$

where q is the draft distribution induced by M_q, and $\alpha(\cdot)$ denotes the speculative verification rule. In standard non-greedy speculative sampling, this acceptance rule uses rejection sampling:

$$\alpha_t = \min\left(1,\, \frac{p(y_t \mid x, y_{<t})}{q(y_t \mid x, y_{<t})}\right), \tag{9}$$

with a target-model correction token sampled when rejection occurs (Leviathan et al., 2023). The accepted prefix length is $a = \max\{r \leq k : A_t = 1,\ \forall t \leq r\}$. In the greedy setting, A_t becomes deterministic: a draft token is accepted if it matches the target model's next-token prediction.

4 Experiments

4.1 Experiment Setup

Models and Datasets. We evaluate JetSpec on Qwen3-8B and Qwen3-30B-A3B, covering both dense and MoE target models (Yang et al., 2025). Qwen3 supports both thinking and non-thinking modes, and we use the non-thinking mode throughout our main evaluation for efficient decoding. For training data, we curate 780K examples from the Nemotron Post-Training Dataset V2 (NVIDIA, 2025), including all available coding and math splits, random samples from STEM and chat splits, and 20K additional examples from CodeAlpaca (Chaudhary, 2023). For regenerated training sequences, we apply the corresponding chat template to each data type and continue generation with the target model. We evaluate on math benchmarks including GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021), and AIME25; coding benchmarks including HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and LiveCodeBench (Jain et al., 2024); and open-ended conversational tasks including MT-Bench (Zheng et al., 2023). All ablation studies are conducted on the math split of Nemotron Post-Training Dataset V2.

Baselines and Training Settings. We compare against EAGLE-3 and DFlash, two strong head-based speculative decoding baselines (Li et al., 2025; Chen et al., 2026). EAGLE-3 uses multi-layer feature fusion for direct token prediction, while DFlash uses a bidirectional block-parallel draft head to generate multiple draft tokens in one pass. DDTree (Ringel & Romano, 2026) constructs draft trees from DFlash's per-position block-diffusion distributions using best-first tree expansion, but uses a pure block-diffusion head trained without loss weighting (Ringel & Romano, 2026). For fair comparison, we train JetSpec and all baselines on the same data mixture using a learning rate of 3×10^{-4} on 8 H100 GPUs. We also ablate DFlash-style loss weighting in Section 4.4; additional training details are provided in Appendix D.

Table 1: Low-budget regime comparison of JetSpec and baselines trained with the same Qwen3-8B model and data recipe. We report results on math, coding, and chat benchmarks using non-thinking mode with a 3072 max tokens. We report end-to-end decoding speedup over standard AR decoding and average accepted length τ on H100 GPU.

Method	Budget	GSM8K		MATH-500		AIME25		HumanEval		MBPP		LCB		MT-Bench	
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Temperature = 0															
EAGLE-3	16	2.24	3.78	2.10	3.61	2.08	3.55	2.18	3.72	1.95	3.31	1.82	3.19	1.91	3.40
	32	2.39	4.03	2.23	3.87	2.22	3.80	2.34	4.00	2.09	3.60	1.96	3.42	2.04	3.62
DFlash	16	4.80	6.01	6.12	7.83	**5.85**	**7.34**	4.14	5.18	3.96	4.98	4.70	6.16	**2.72**	4.03
	32	4.21	5.27	5.39	6.89	5.15	6.38	3.63	4.53	3.51	4.39	3.99	5.23	2.48	3.61
JetSpec	16	4.80	6.00	6.06	7.75	5.78	7.23	4.26	5.32	3.97	4.96	4.77	6.24	2.68	3.98
	32	**4.89**	**6.14**	**6.35**	**8.23**	5.75	7.25	**4.29**	**5.35**	**4.03**	**5.03**	**4.86**	**6.48**	2.67	**4.09**
Temperature = 1															
EAGLE-3	16	2.16	3.65	2.01	3.48	1.89	3.24	2.02	3.58	1.89	3.22	1.81	3.09	1.85	3.24
	32	2.26	3.90	2.12	3.74	2.01	3.49	2.19	3.86	2.02	3.50	1.91	3.31	1.90	3.42
DFlash	16	4.32	5.44	4.87	6.30	3.55	4.69	3.56	4.45	3.52	4.43	4.38	5.70	**2.41**	3.52
	32	3.85	4.84	4.29	5.55	3.44	4.48	3.22	4.00	3.17	3.97	3.74	4.81	2.23	3.20
JetSpec	16	4.32	5.44	4.77	6.18	3.48	4.54	3.70	4.63	3.52	4.41	4.41	5.74	**2.41**	3.53
	32	**4.40**	**5.58**	**4.91**	**6.44**	**3.64**	**4.76**	**3.72**	**4.66**	**3.60**	**4.51**	**4.53**	**5.93**	2.40	**3.54**

Implementation Details. For offline inference evaluation, we run experiments on 8 H100 or B200 GPUs. Unless otherwise specified, results in Table 2 use optimized Triton kernels for decoding. For serving-engine evaluation, we integrate JetSpec into vLLM (Appendix E), part of which involves tree verification with a custom SM90 paged FlashAttention kernel using NVIDIA CuTe DSL, extending the paged attention forward path with shared-memory tree-mask staging and tree-tail masking.

4.2 Results

We evaluate JetSpec under both greedy and non-greedy decoding settings in compliance with Equation 9. Evaluation results and comparisons with baselines are summarized in Table 1 and Table 2, and it shows JetSpec consistently outperforms tree-based baselines across all benchmarks.

Low-budget Regime. Compared with the autoregressive drafting baseline EAGLE-3, JetSpec achieves substantially higher speedup at every reported budget by avoiding EAGLE-3's higher sequential drafting overhead. At budget 16, DFlash and JetSpec perform nearly the same, suggesting that a short linear draft is already sufficient to cover many high-probability continuations. However, when increasing the budget to 32, JetSpec brings slight improvement, while DFlash often saturates or degrades. This indicates that JetSpec converts the additional draft budget into accepted tokens more effectively.

High-budget Regime. In Table 2, as the draft budget increases, JetSpec shows a larger gain, e.g. improving from roughly 4–$6\times$ to 7–$10\times$ speedup on math and coding benchmarks. Whereas DDTree scales more modestly to up to $9\times$, showing causal tree drafting yields higher acceptance than branch-agnostic tree construction. And detailed qualitative example in Appendix A.2 on how causality helps. Under temperature 1, JetSpec remains effective, showing that the benefit of causal tree drafting remains robust under nongreedy decoding.

4.3 System Performance

We evaluate JetSpec in an end-to-end serving setting by integrating tree drafting and verification into vLLM. Serving performance depends not only on accepted length, but also on how the tree budget interacts with batching, verification overhead, memory traffic, and GPU occupancy. Larger budgets can reduce the number of target-model verification rounds, but also introduce higher draft-head and tree-verification cost, making the optimal budget load-dependent.

Table 2: High-budget comparison on Qwen3-8B with at least 64 draft tokens. EAGLE-3 uses tree mode with max depth 8; larger budgets give minimal or worse gains due to training mismatch.

Method	Budget	GSM8K		MATH-500		AIME25		HumanEval		MBPP		LCB		MT-Bench	
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Temperature = 0															
EAGLE-3	64	2.53	4.31	2.36	4.13	2.35	4.04	2.49	4.26	2.22	3.81	2.09	3.62	2.19	3.88
DDTree	64	5.63	6.18	6.51	7.16	6.40	6.96	5.08	5.57	4.99	5.49	5.47	6.06	3.74	4.51
	128	6.63	7.31	8.27	9.19	7.93	8.66	5.93	6.52	5.70	6.28	6.42	7.25	4.12	5.14
	256	7.04	7.77	8.78	9.81	8.33	9.24	6.31	6.96	6.09	6.70	6.75	7.72	4.26	5.41
JetSpec	64	5.98	6.56	6.76	7.42	6.47	7.00	5.53	6.06	5.34	5.88	5.95	6.59	3.97	4.77
	128	7.34	8.05	8.93	9.95	8.26	9.10	6.66	7.28	6.31	6.95	7.29	8.21	4.37	5.52
	256	**7.82**	**8.62**	**9.64**	**10.76**	**8.78**	**9.82**	**7.12**	**7.78**	**6.73**	**7.43**	**7.67**	**8.79**	**4.58**	**5.94**
Temperature = 1															
EAGLE-3	64	2.35	4.17	2.23	4.00	2.11	3.73	2.35	4.13	2.13	3.70	1.99	3.49	1.96	3.63
DDTree	64	5.28	5.77	5.68	6.26	4.94	5.46	4.65	5.09	4.65	5.11	5.28	5.82	3.50	4.22
	128	6.11	6.72	6.79	7.60	5.40	5.98	5.29	5.76	5.22	5.75	5.99	6.71	3.66	4.59
	256	6.41	7.17	7.10	8.13	5.26	6.20	5.43	6.03	5.49	6.12	6.26	7.17	3.81	4.88
JetSpec	64	5.63	6.19	5.97	6.60	4.95	5.48	5.02	5.52	5.00	5.51	5.76	6.38	3.63	4.37
	128	6.76	7.49	7.39	8.36	5.76	6.49	5.82	6.39	5.78	6.39	6.84	7.74	3.99	5.01
	256	**7.16**	**8.03**	**7.83**	**9.01**	**5.94**	**7.06**	**6.19**	**6.85**	**6.11**	**6.83**	**7.25**	**8.29**	**4.06**	**5.22**

Budget Allocation. Table 8 shows that larger tree budgets are most useful at small batch sizes. At batch size 1, increasing the budget from 16 to 128 raises throughput from 224.0 to 553.3 TPS and speedup from 1.75× to 4.33×. At batch size 16, a moderate budget of 32 performs best at 3.81×, while budget 128 drops to 2.80×. This illustrates that larger budgets are most beneficial at low load, whereas heavier batching favors more moderate budgets.

Practical implication. These results suggest that tree budgets should be selected according to serving load. Large budgets are preferable in low- to moderate-load regimes, where spare GPU capacity can be used to reduce decoding iterations. Under heavier load, smaller or moderate budgets may be more efficient by reducing per-step overhead and preserving batching efficiency. We evaluate static budget policies in this work and leave dynamic serving-time budget scheduling to future work.

4.4 Ablation Studies

Unless noted, all ablations in this section are evaluated on $4\times$B200 with block size 16, FlashAttention-2 for the target's normal forward, Triton tree-attention, and the accumulated draft log-probability heuristic (Equation 7) for tree construction at node budget 255. Reported average accepted length τ are independent of hardware under lossless verification.

4.4.1 Training Settings

Learning Rates. Table 9 reports a grid search over learning rates from 5×10^{-5} to 1×10^{-3}. Very small learning rates underperform and performance plateaus at 3×10^{-4} (peak speedup $8.30\times$), with 6×10^{-4} and 1×10^{-3} within $\sim 2\%$ of the peak.

Loss. Table 10 compares SFT, forward-KL distillation, and reverse-KL distillation across four math benchmarks. SFT and forward-KL distillation match within $\sim 3\%$ at every dataset, while reverse-KL distillation causes a 36–46% relative drop in comparison with forward-KL's, indicating the mode-seeking objective concentrates draft probability mass too aggressively for tree drafting.

Model Generalizability. Table 3 compares JetSpec and DDTree trained on Qwen3-30B-A3B, a MoE target model, with the same 800k data recipe. JetSpec maintains a competitively higher speedups across benchmarks, showing that causal tree drafting generalizes beyond dense Qwen3-8B.

Training Data. Table 11 studies the effect of using different training data. Regenerated target-model sequences provide the strongest performance, confirming that matching the draft head to the target model's own generations is important for maximizing accepted length. In contrast, JetSpec-Corpus

Table 3: Model generalizability: JetSpec vs. DDTree on Qwen3-30B-A3B (MoE target), both trained with SFT on the same 800K-example data mixture as our Qwen3-8B main results. Each cell reports speedup / average accepted length τ at temperature 0 with tree budget 256.

Method	GSM8K	MATH-500	AIME25	HumanEval	MBPP	LCB	MT-Bench
DDTree	7.26 / 7.93	8.61 / 9.49	9.01 / 9.71	6.18 / 6.76	6.39 / 7.06	7.40 / 8.31	4.26 / 5.35
JetSpec	7.40 / 8.18	9.45 / 10.65	9.35 / 10.28	6.51 / 7.23	6.53 / 7.29	7.47 / 8.62	4.33 / 5.59

trains directly on corpus sequences, it underperforms fully regenerated JetSpec, but still achieves consistent speedups across all benchmarks, suggesting that causal parallel drafting can potentially be incorporated during mid-training or even pretraining.

4.4.2 Tree Drafting with Diffusion Head

Table 4 compares causal and diffusion heads under different choices of γ, the parameter that controls how aggressively the DFlash training objective downweights per-position loss at positions far from each anchor token. Specifically, position i within a block contributes to the training loss with weight $w_i = \exp(-\max(i - i_{\text{anchor}}, 0)/\gamma)$, where $\gamma = 0$ is interpreted as the limit of uniform per-position weighting (no decay). The causal head is robust across γ, while the diffusion head is sensitive to γ and collapses at the endpoints (from $8.36\times$ at $\gamma = 7$ to $5.46\times$ at $\gamma = 0$ and $6.17\times$ at $\gamma = 15$).

Table 4: Architecture and γ ablation on MATH-500 with JetSpec at LR 3×10^{-4} with forward-KL distillation. We report speedup and average accepted length τ.

Head	$\gamma = 0$		$\gamma = 3$		$\gamma = 7$		$\gamma = 15$	
	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Causal	8.29	9.81	8.50	10.00	8.40	9.99	8.41	9.96
Diffusion	5.46	6.45	8.16	9.65	8.36	9.72	6.17	7.19

A failure example. On MATH-500 prompt 0 at decode step 0, the diffusion-head draft ($\gamma = 0$) ranks " `given told that`" first in its tree, with cumulative draft surrogate $\sum_i \log r_i = -3.76$ but cumulative target conditional $\sum_i \log p(y_i \mid x, \pi_{<v_i}) = -63.32$ nats: the branch combines two mutually exclusive openers, exemplifying the failure mode discussed in Section 3.2. The actually-coherent " `are given that the`" (target joint -0.08) sits at rank 3. The causal-head draft on the same prompt ranks " `are told that`" first, with surrogate $\approx$ target joint (gap -0.34); tree verification accepts 6 tokens from the causal-head tree but only 4 from the diffusion-head tree (visualized in Figure 4 of Appendix A).

Aggregate generalization. Across MATH-500 prompts 0–49, the diffusion-head's rank-1 gap exceeds the causal-head's on 92% of prompts, with a median $5\times$ larger ($+62.81$ vs. $+12.36$ nats); the mean per-step accepted-token ratio is $1.95\times$, closely matching the macroscopic $1.52\times$ speedup ratio in Table 4. Full branch-level tables and the rank-1 gap distribution are in Appendix A.

5 Conclusion

We presented JetSpec, a causal parallel tree drafting framework for speculative decoding that jointly addresses the two factors limiting end-to-end speedup: drafting latency and accepted length. By causally coupling draftrepresentations while generating all depth-wise candidate distributions in a single low-cost pass, JetSpec makes tree construction better aligned with the target model's autoregressive factorization. Across dense and MoE Qwen3 models and vLLM serving settings, JetSpec consistently turns larger draft budgets into higher wall-clock speedup, for MATH-500, reaching $9.64\times$ on Qwen3-8B, $9.45\times$ on Qwen3-30B-A3B, and up to $4.33\times$ serving throughput speedup on a single H100. These results show that causal coupling across draft positions improves long-budget speculative drafting while retaining one-pass efficiency.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on technical formulations and qualitative data analysis, we also used AI to help with method implementations. We have not used generative AI tools to generate synthetic datasets, develop theoretical models, assist in writing proofs, propose hypotheses, translate content, clean or reformat datasets, or interpret experimental results. Additionally, we used generative AI tools to edit and refine manuscript text for clarity and readability, suggest improvements to presentation and structure, and assist with literature search and verification. We reviewed all AI-assisted work and manually verified technical claims, citations, and suggested revisions against the original sources and our own experiments. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the proposed method, training procedure, and system implementation to facilitate reproducibility. Appendix B specifies the parallel tree-drafting algorithm and scoring procedure; Appendix D reports the draft-head architecture, target-feature extraction, training data construction, optimization settings, and loss configurations; Appendix E describes the vLLM integration and optimized tree-verification implementation. We also provide the experimental code in the supplementary material.

ETHICS STATEMENT

All authors have read and adhere to the ICLR Code of Ethics. This work does not involve human subjects, sensitive personal data, or experiments with the potential to cause harm. No confidential or proprietary data were used. The methods and experiments were conducted in accordance with principles of research integrity, fairness, and transparency. All conclusions are the sole responsibility of the authors.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes, 2024. URL https://arxiv.org/abs/2306.13649.

StepFun AI. Step 3.5 flash: Open frontier-level intelligence with 11b active parameters. *arXiv preprint arXiv:2602.10604*, 2026. URL https://arxiv.org/abs/2602.10604.

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

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

Sahil Chaudhary. Code Alpaca: An instruction-following LLaMA model trained on code generation instructions. https://github.com/sahil280114/codealpaca, 2023.

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

Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026. URL https://arxiv.org/abs/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, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. URL https://arxiv.org/abs/2412.19437.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026. doi: 10.48550/arXiv.2606.19348. URL https://arxiv.org/abs/2606.19348.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks?, 2025. URL https://arxiv.org/abs/2509.16941.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.681. URL https://aclanthology.org/2024.acl-long.681/.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 14060–14079. PMLR, 2024.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024. URL https://arxiv.org/abs/2404.19737.

Raghavv Goel, Mukul Gagrani, Wonseok Jeon, Junyoung Park, Mingu Lee, and Christopher Lott. Direct alignment of draft model for speculative decoding with chat-fine-tuned llms. *arXiv preprint arXiv:2403.00858*, 2024.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *International Conference on Learning Representations*, 2024.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1582–1595, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks*, 2021.

Lanxiang Hu, Siqi Kou, Yichao Fu, Samyam Rajbhandari, Tajana Rosing, Yuxiong He, Zhijie Deng, and Hao Zhang. Fast and accurate causal parallel decoding using jacobi forcing. *arXiv preprint arXiv:2512.14681*, 2025a.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. SAM decoding: Speculative decoding via suffix automaton. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12187–12204, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.595. URL https://aclanthology.org/2025.acl-long.595/.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. URL https://arxiv.org/abs/2403.07974.

Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. CLLMs: Consistency large language models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023. URL https://arxiv.org/abs/2211.17192.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024a. URL https://arxiv.org/abs/2401.15077.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*, 2024b. URL https://arxiv.org/abs/2406.16858.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025. URL https://arxiv.org/abs/2503.01840.

Yihao Liang, Ze Wang, Hao Chen, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Emad Barsoum, Zicheng Liu, and Niraj K. Jha. CD4LM: Consistency distillation and adaptive decoding for diffusion language models. *arXiv preprint arXiv:2601.02236*, 2026.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 31131–31146. PMLR, 2024.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

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

NVIDIA. Nemotron post-training dataset v2. https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v2, 2025. Dataset.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. Suffixdecoding: Extreme speculative decoding for emerging ai applications. *arXiv preprint arXiv:2411.04975*, 2024. URL https://arxiv.org/abs/2411.04975.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Yu-Yang Qian, Junda Su, Lanxiang Hu, Peiyuan Zhang, Zhijie Deng, Peng Zhao, and Hao Zhang. d3LLM: Ultra-fast diffusion LLM using pseudo-trajectory distillation. *arXiv preprint arXiv:2601.07568*, 2026.

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

Apoorv Saxena. Prompt lookup decoding. `https://github.com/apoorvumang/prompt-lookup-decoding`, 2023. GitHub repository.

Shwetha Somasundaram, Anirudh Phukan, and Apoorv Saxena. Pld+: Accelerating llm inference by leveraging language model artifacts. *arXiv preprint arXiv:2412.01447*, 2024. URL `https://arxiv.org/abs/2412.01447`.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf`.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm. *arXiv preprint arXiv:2509.26328*, 2025. URL `https://arxiv.org/abs/2509.26328`.

LLM-Core Xiaomi. Mimo-v2-flash technical report. *arXiv preprint arXiv:2601.02780*, 2026. URL `https://arxiv.org/abs/2601.02780`.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R. Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=roNSXZpUDN`.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*, 2025. URL `https://arxiv.org/abs/2504.12216`.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

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

Dawei Zhu, Xiyu Wei, Guangxiang Zhao, Wenhao Wu, Haosheng Zou, Junfeng Ran, XWang, Lin Sun, Xiangzheng Zhang, and Sujian Li. Chain-of-thought matters: Improving long-context language models with reasoning path supervision. In *Findings of the Association for Computational*

Linguistics: EMNLP 2025, pp. 3197–3211, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.170. URL https://aclanthology.org/2025.findings-emnlp.170/.

A TREE DRAFTING QUALITY ANALYSIS

This appendix provides the full data behind the rank-1 case study and the $N = 50$ aggregate statistics summarized in Section 4.4.2. Figure 4 visualizes the rank-1 + rank-3 contrast for both heads on MATH-500 prompt #0, decode step 0 (root token "`We`"); the full top-5 branches and the rank-1 gap distribution follow.

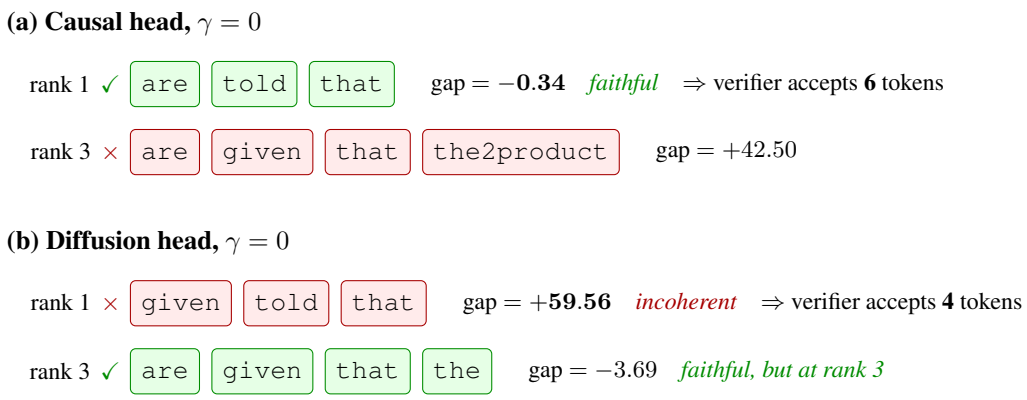


Figure 4: **Tree-quality failure mode at MATH-500 prompt #0, decode step 0.** Both heads draft from the same prefix (last token "`We`"). The causal head's rank-1 branch ("` are told that`") is faithful: target joint $\Sigma \log p \approx \Sigma \log r$, so tree verification walks 6 tokens along it. The diffusion head's rank-1 branch ("` given told that`") is incoherent (target joint $\Sigma \log p = -63.32$ nats, i.e. probability $\approx e^{-63}$) because its branch-agnostic per-position predictor composes "` given`" (depth 1) and "` told`" (depth 2) independently in the surrogate, even though no real continuation places these two words consecutively. The actually-coherent "` are given that the`" is in the diffusion tree but only at rank 3; the surrogate fails to promote it, and the verifier accepts only 4 tokens.

A.1 SETUP

We modify the speculative decoding loop to dump the constructed tree at a chosen decode step. For every node v we record the cumulative draft surrogate $\sum_{i=1}^{\text{depth}(v)} \log r_i(y_{v_i} \mid x)$ and the cumulative target conditional $\sum_{i=1}^{\text{depth}(v)} \log p(y_{v_i} \mid x, \pi_{<v_i})$, along with the tree-verifier's accepted path. The two drafts compared are a causal head and a diffusion head, both trained at LR 3×10^{-4} with $\gamma = 0$, evaluated on 4×B200 with block size $N = 16$ (also the maximum draft depth), FlashAttention-2, and Triton tree-attention. Both heads share the prefill and produce N parallel position-wise distributions $r_1, \ldots, r_N$; tree construction uses the cumulative-log-probability heuristic with branching width $W = 7$ and node budget $B = 255$.

A.2 TOP-5 BRANCHES AT THE CANONICAL EXAMPLE

Table 5 extends Figure 4 to the full top-5 branches of each head's tree at MATH-500 prompt #0, decode step 0 (root token "`We`"). The pattern reported in the main text repeats throughout the tree. For the diffusion head, top-2 and top-4 both combine "` given`" at depth 1 and "` told`" at depth 2 with target joints below -50 nats; only rank-3 (target joint -0.08) is coherent. For the causal head, the rank-1 branch ("` are told that`") is faithful (gap -0.34); ranks 2–5 all share the off-argmax depth-2 token "` given`" (vs. rank-1's argmax "` told`") and degrade at depths 4–5 (e.g. "`the2product`" at rank 3, gap $+42.50$), consistent with the off-argmax inheritance described in §A.4: each branch's depth-d marginal was anchored to the argmax extension at depth $d-1$, so off-argmax branches inherit a conditioning context that does not match their own ancestor token.

Table 5: Top-5 branches of each head's tree at MATH-500 prompt 0, decode step 0. Both drafts run on the identical prefix with the same target. $\Sigma \log r$ and $\Sigma \log p$ in nats; Δ = surrogate − target.

Architecture	Rank	Branch text	$\Sigma \log r$	$\Sigma \log p$	Δ
Causal γ=0	1	`" are told that"`	−3.88	−3.54	−0.34
	2	`" are given that the product **"`	−3.96	−11.68	+7.73
	3	`" are given that the2product"`	−3.96	−46.46	+42.50
	4	`" are given that the2 of"`	−3.96	−49.71	+45.75
	5	`" are given:\n\n the **"`	−4.01	−40.59	+36.58
Diffusion γ=0	1	`" given told that **"`	−3.76	−63.32	+59.56
	2	`" given told:\n\n the"`	−3.77	−90.78	+87.02
	3	`" are given that the"`	−3.77	−0.08	−3.69
	4	`" told given that"`	−3.84	−50.94	+47.10
	5	`" given told that:\n\n product"`	−3.87	−96.44	+92.57

A.3 Gap Distribution over 50 Prompts

Table 6 reports the rank-1 gap distribution across MATH-500 prompts 0–49 for both heads at $\gamma = 0$ and at $\gamma = 7$ (DFlash's best macroscopic loss-weighting setting, Table 4). At $\gamma = 0$, the diffusion head exhibits a dramatic upper-tail failure: 26% of prompts have rank-1 gap $\geq +80$ nats (joint target probability $< 10^{-35}$), versus 0% for the causal head; faithful rank-1 (gap $< +5$ nats) appears in only 6% of diffusion prompts versus 42% for causal. The macroscopic effect is direct: mean accepted length is 4.84 tokens for diffusion vs 9.46 for causal at $\gamma = 0$. At $\gamma = 7$, the diffusion head's tree quality recovers substantially — its extreme tail drops to 4% and faithful rank-1 rises to 26% — with mean acceptance reaching 9.42 tokens. It does not, however, match the causal head, which retains the lower extreme tail (2%) and higher mean acceptance (9.64) at $\gamma = 7$. The takeaway is that the causal head's rank-1 faithfulness is robust across γ, whereas the diffusion head depends on loss-weighting tuning to recover from the $\gamma = 0$ failure mode. JetSpec 's contribution is the structural robustness this provides: the causal mask between depths obviates the need to tune γ.

Table 6: Distribution of rank-1 surrogate–target gaps across MATH-500 prompts 0–49 (each at the prompt's natural step-0 tree). $\gamma = 7$ is DFlash's best macroscopic loss-weighting setting (Table 4). Mean accepted length per step is reported in the bottom row for context.

Rank-1 gap (nats)	$\gamma = 0$		$\gamma = 7$	
	Causal	Diffusion	Causal	Diffusion
$< +5$ (faithful)	42% (21/50)	6% (3/50)	32% (16/50)	26% (13/50)
$[+5, +20)$	12% (6/50)	0% (0/50)	6% (3/50)	14% (7/50)
$[+20, +40)$	24% (12/50)	4% (2/50)	32% (16/50)	20% (10/50)
$[+40, +80)$	22% (11/50)	64% (32/50)	28% (14/50)	36% (18/50)
$\geq +80$ (extreme)	0% (0/50)	26% (13/50)	2% (1/50)	4% (2/50)
Median gap (nats)	+12.36	+62.81	+27.87	+32.69
Mean gap (nats)	+20.46	+63.25	+26.58	+32.07
Mean accepted length	9.46	4.84	9.64	9.42

A.4 Why the Failure Mode Is Structural

The diffusion head produces all N position-wise distributions $r_1, \ldots, r_N$ from a single shared hidden state with no causal mask between depths, so r_2 does not condition on y_1. When r_1 and r_2 both place high mass on tokens that fit a given depth marginally but cannot follow each other syntactically (e.g. " `given`" at depth 1 and " `told`" at depth 2), the surrogate $r_1(y_1) \cdot r_2(y_2)$ promotes their composition into the rank-1 branch. The causal head's parallel marginals are produced with causal masking between depths, so r_2 is anchored to the model's own argmax at depth 1; off-argmax branches in the heap still inherit this anchored r_2, but the rank-1 branch (the argmax-aligned one, which dominates verifier behavior) receives a faithful surrogate score. This anchoring is what closes

the gap between q_{sur} and the target's branch-conditional joint at the head of the tree, and it is what allows the causal head to remain robust across γ: the rank-1 alignment does not depend on a loss-weighting prior over depths.

B TREE-DRAFTING ALGORITHM DETAILS

Algorithm 1 describes how JETSPEC constructs a candidate tree under a fixed node budget. Starting from the root prefix x, the algorithm maintains a priority queue of expandable nodes ranked by a branch scoring function $\mathrm{Score}(\cdot)$. At each iteration, it expands the highest-scoring node by adding up to W children at the next depth, unless the node has already reached the maximum draft depth N. Each new child is inserted back into the queue with its updated path score, and the process continues until the total node budget B is exhausted. The returned tree $\mathcal{T}(x)$ therefore contains the highest-scoring candidate branches according to the chosen scoring rule.

Algorithm 1 Parallel Tree Drafting

Require: Prefix x, maximum draft depth N, branching width W, node budget B, scoring function $\mathrm{Score}(\cdot)$
Ensure: Candidate tree $\mathcal{T}(x)$
Initialize tree $\mathcal{T}$ with root node v_0
Initialize priority queue $\mathcal{Q} \leftarrow \{(v_0, \mathrm{Score}(\pi(v_0)))\}$
while $|\mathcal{V}_{\mathcal{T}}| < B$ and $\mathcal{Q} \neq \emptyset$ **do**
    Pop the highest-scoring node v from $\mathcal{Q}$
    if $\mathrm{depth}(v) = N$ **then**
        continue
    end if
    Obtain up to W candidate children $\mathcal{C}(v)$ for the next depth
    for $y \in \mathcal{C}(v)$ **do**
        if $|\mathcal{V}_{\mathcal{T}}| = B$ **then**
            break
        end if
        Add child node u with token $y_u = y$ and parent v to $\mathcal{T}$
        Compute $s_u \leftarrow \mathrm{Score}(\pi(u))$
        Push (u, s_u) into $\mathcal{Q}$
    end for
end while
return $\mathcal{T}(x) = \{\pi(v) \mid v \in \mathcal{V}_{\mathcal{T}}\}$

C TREE-DRAFTING ALGORITHM ABLATION

We compare three tree-construction scoring algorithms at the production setting: cumulative draft log-probability (`accum_logp`, our default), entropy-guided (per-depth entropy alone), and hybrid (cumulative log-probability plus α-weighted per-depth entropy). Results are summarized in Table 7.

Cumulative log-probability scoring. `accum_logp` achieves $8.15\times$ speedup (average accepted length $\tau = 9.81$) and is our production default. It expands the tree by best-first heap priority on cumulative draft log-probability along each branch, prioritizing high-likelihood continuations under the budget.

Entropy-only collapse. Entropy-guided scoring (frontier priority by per-depth marginal entropy alone, with no cumulative-log-prob signal) drops to $4.76\times$ (-42% relative to `accum_logp`), confirming that per-depth entropy alone is insufficient to identify high-acceptance branches.

Hybrid scoring with increasing α. Hybrid scoring uses $\sum_i \log r_i + \alpha \cdot H_i$, with per-depth entropy H_i. At $\alpha \leq 1$, hybrid converges within $\sim 1.5\%$ of `accum_logp` (8.15–$8.27\times$); increasing α monotonically degrades acceptance, reaching $7.42\times$ at α=8. The $\alpha \to 0$ limit reduces to pure

cumulative log-probability (consistent with `accum_logp` sitting on the same plateau as low-α hybrid); the $\alpha \to \infty$ limit corresponds to entropy-only scoring (consistent with the $4.76\times$ collapse).

Table 7: Tree-construction algorithm ablation on MATH-500 ($n = 500$) with JetSpec at the production setting (causal head, LR 3×10^{-4}, Forward-KL distillation, $\gamma = 0$). Hybrid scoring is $\sum_i \log r_i + \alpha \cdot H_i$ with per-depth entropy H_i. We report speedup and average accepted length τ.

	Speedup	τ
Algorithm		
Accum. log-prob (default)	**8.15**	9.81
Entropy-guided	4.76	5.52
Hybrid (log-prob + α· entropy)		
$\alpha = 0.25$	8.27	9.81
$\alpha = 0.5$	8.22	9.78
$\alpha = 1.0$	8.15	9.78
$\alpha = 2.0$	7.98	9.63
$\alpha = 4.0$	7.75	9.27
$\alpha = 8.0$	7.42	9.00

D Training Details

For both DFlash and JetSpec , we perform a learning-rate sweep from 1×10^{-4} to 1×10^{-3} with five settings. We find that 3×10^{-4} and 6×10^{-4} generally perform best, with different tasks favoring different choices. Since their overall performance is comparable, we report results with 3×10^{-4} by default. All draft-head training runs are conducted on 8 H100 GPUs with a micro batch of 2.

For a fair comparison with DFlash, the JetSpec draft head is trained with block size 16, corresponding to a maximum tree depth of 16, and uses fused hidden features extracted from the frozen target model. During training, each block keeps the first token as the anchor and replaces the remaining positions with mask-token embeddings. The causal head predicts all masked future tokens in parallel, while each position can attend only to the prefix and earlier positions within the same block, ensuring that the draft distribution follows the autoregressive order. We sample random anchor positions from each sequence, use up to 512 anchors per example. An example of the attention mask with 3 anchors for 3 blocks is provided in Figure 5.

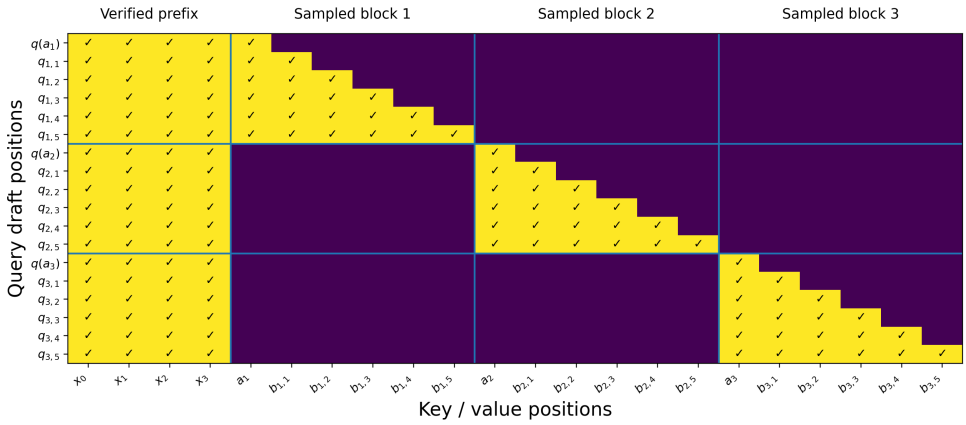


Figure 5: Causal attention mask used for training with multiple sampled blocks. Each query can attend to the full verified prefix and to the anchor plus earlier positions within its own block, but cannot attend to future positions or positions from other sampled blocks.

JetSpec reuses intermediate representations from the frozen target model as draft-head context. For Qwen3-8B, we extract hidden states from target layers $\{1, 9, 17, 25, 33\}$ out of the 36-layer target model, concatenate them along the channel dimension, and project the resulting $5d$ feature back to

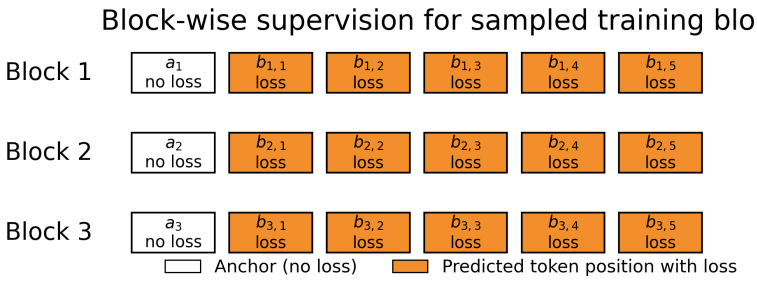


Figure 6: Each sampled block includes an anchor position and multiple future token positions. The anchor is retained as block context and excluded from the loss, while loss is applied only to future token positions within each block.

hidden size $d = 4096$ through a bias-free linear layer followed by RMSNorm. The draft head is implemented as a lightweight Qwen3-style decoder with 5 layers, 32 attention heads, 8 KV heads, head dimension 128, and MLP intermediate size 12288. In each draft layer, the projected target feature is injected as contextual key/value states and concatenated with the draft-token hidden states, allowing the causal draft head to condition on rich target-model features while keeping the target model frozen.

For ablations, we consider DFlash-style exponential depth weighting. Since depth weighting can implicitly bias the bidirectional DFlash head toward left-to-right prediction, in main results reported in Table 1 and Table 2, we remove it when isolating the effect of causal masking versus the native block-diffusion head design. For distillation runs, teacher logits are obtained from the frozen target model and aligned with the student prediction positions; the draft head is trained with a temperature-scaled soft-label distillation loss.

E vLLM Implementation Details

We implement causal parallel tree drafting inside vLLM. The draft head uses causal attention, while its candidates are organized as a speculative tree rather than a linear block.

Given the target hidden states at the current decoding step, the proposer produces draft logits for a block of future positions. For each position, it extracts the top-k tokens and log-probabilities, where k is the tree width, and constructs a speculative tree under a fixed node budget. Our implementation supports several expansion scores, including cumulative log-probability, entropy-guided scoring, and hybrid scoring, as discussed in Appendix C. We use cumulative log-probability in our experiments. The resulting tree stores token ids, parent indices, and node depths, with the root corresponding to the already-sampled next token.

During verification, the tree metadata is passed to the target model as parent indices and depths for all speculative nodes. The target verifies all tree nodes in one forward pass using a tree attention mask. To reduce verification overhead, we implement an optimized fused paged tree-attention kernel: it builds the ancestor relation for speculative nodes, applies the tree mask inside attention, and verifies all candidates without materializing a dense per-request mask. Acceptance is computed by comparing each node token with the target model's greedy prediction at its parent, then selecting the deepest fully accepted root-to-node path. The accepted path is committed, and the target prediction at the final accepted node is used as the correction token.

Table 8 reports an example end-to-end vLLM run on MATH-500 with a single H100 GPU. The results show how tree budget interacts with serving batch size: larger budgets improve low-batch latency by reducing verification rounds, while their relative gain decreases at larger batch sizes as verification and memory pressure become more prominent.

Table 8: vLLM serving performance of JETSPEC on Math-500 with Qwen3-8B on a single H100 GPU, evaluated across batch sizes and tree budgets (in parentheses). Each setting reports throughput in tokens/s (TPS), with speedup over AR decoding shown in parentheses.

Batch Size	AR	JetSpec (16)	JetSpec (32)	JetSpec (64)	JetSpec (128)
1	127.8	224.0 (1.75×)	312.0 (2.44×)	447.3 (3.50×)	553.3 (4.33×)
2	163.3	266.2 (1.63×)	360.3 (2.21×)	516.0 (3.16×)	621.6 (3.81×)
4	203.8	433.6 (2.13×)	534.2 (2.62×)	664.2 (3.26×)	742.9 (3.64×)
8	246.2	679.3 (2.76×)	839.3 (3.41×)	859.3 (3.49×)	803.5 (3.26×)
16	287.3	891.8 (3.10×)	1094.6 (3.81×)	995.8 (3.47×)	803.1 (2.80×)

F ABLATION STUDY DETAILS

We provide detailed ablation results for the training and generalization settings summarized in Section 4. Unless otherwise specified, ablations are conducted on the Nemotron Post-Training Dataset V2 math split, using Qwen3-8B with block size 16, tree node budget 255, accumulated draft log-probability for tree construction, and the same evaluation protocol as the main experiments.

Table 9: Learning-rate ablation with JETSPEC and without loss weighting training ($\gamma = 0$). See Section 4.4.2 for γ's definition and ablations. We report speedup and average accepted length τ.

LR	GSM8K				MATH-500			
	SFT		Forward KL		SFT		Forward KL	
	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
5×10^{-5}	4.69	5.46	4.76	5.58	7.27	8.70	7.29	8.71
1×10^{-4}	5.37	6.22	5.45	6.32	7.89	9.34	7.89	9.39
3×10^{-4}	5.79	6.78	5.92	6.87	8.30	9.80	8.29	9.81
6×10^{-4}	5.87	6.74	5.79	6.78	8.23	9.73	8.14	9.70
1×10^{-3}	5.75	6.64	5.73	6.70	8.17	9.66	8.15	9.66

Learning Rate. Table 9 studies the effect of draft-head learning rate. Very small learning rates underfit the drafter, while performance plateaus around 3×10^{-4}; larger rates remain competitive but do not provide consistent additional gains.

Table 10: Loss-objective ablation with JETSPEC at LR 6×10^{-4} and $\gamma = 0$. We report speedup and average accepted length τ. All cells use checkpoints from a single training pipeline trained on math-only data ($\sim$3 epochs); we report on math benchmarks to keep the comparison in-distribution.

Dataset	SFT		Forward-KL Distill		Reverse-KL Distill	
	Speedup	τ	Speedup	τ	Speedup	τ
GSM8K	5.96	6.93	6.11	7.09	3.29	3.78
MATH-500	8.42	9.98	8.46	10.01	5.25	6.59
AIME25	7.51	9.51	7.56	9.40	4.76	6.00
AIME24	8.25	9.93	8.00	9.70	5.13	6.43

Training Objective. Table 10 compares SFT, forward-KL distillation, and reverse-KL distillation. SFT and forward-KL perform similarly, while reverse-KL substantially degrades performance, suggesting that mode-seeking distillation is poorly matched to budgeted tree drafting, where preserving multiple plausible continuations is important.

Model Generalizability. Table 3 evaluates whether JetSpec generalizes beyond dense Qwen3-8B. On the Qwen3-30B-A3B MoE target model, JetSpec continues to outperform DDTree under the same training recipe, showing that causal parallel tree drafting is not specific to a single model architecture.

Table 11: Training-data ablation: JETSPEC vs. JETSPEC-Corpus, both trained with SFT on the same 800K-example data mixture (Qwen3-8B target). JetSpec uses model-regenerated continuations as supervision targets; JetSpec-Corpus uses the original training corpus. Each cell reports speedup / average accepted length τ at temperature 0.

Method	Budget	GSM8K	AIME25	HumanEval	MBPP	LCB	MT-Bench
JETSPEC	16	4.80 / 6.00	5.78 / 7.23	4.26 / 5.32	3.97 / 4.96	4.77 / 6.24	2.68 / 3.98
JETSPEC-Corpus	16	2.06 / 2.54	2.41 / 2.99	2.11 / 2.61	1.93 / 2.39	2.75 / 3.45	1.54 / 2.00
JETSPEC	64	5.98 / 6.56	6.47 / 7.00	5.53 / 6.06	5.34 / 5.88	5.95 / 6.59	3.97 / 4.77
JETSPEC-Corpus	64	2.57 / 2.78	2.70 / 2.89	2.74 / 2.97	2.60 / 2.83	3.43 / 3.64	2.28 / 2.40
JETSPEC	256	7.82 / 8.62	8.78 / 9.82	7.12 / 7.78	6.73 / 7.43	7.67 / 8.79	4.58 / 5.94
JETSPEC-Corpus	256	3.36 / 3.65	3.66 / 4.06	3.53 / 3.82	3.27 / 3.58	4.42 / 4.86	2.63 / 2.98

Training Data. Table 11 compares regenerated target-model sequences with direct corpus training. Regenerated data gives the strongest performance by better matching the target model's own generation distribution. Corpus-trained JetSpec is weaker but still effective.

G EMPIRICAL PER-TOKEN DRAFTING COST ON MODERN HARDWARE

Figure 2 shows that speculative decoding scales better with draft length when the per-token drafting cost is low. In this appendix, we measure this cost for a practical DFlash-style draft head on a single H200 NVL GPU. We use N to denote the draft depth, i.e., the number of draft tokens proposed in one block. We sweep context length and draft depth with request batch size fixed to one, and report the per-draft-token cost ratio corresponding to the coefficient c in Eq. equation 1.

Setup. We profile a DFlash configuration using Qwen3-8B as the target model and `z-lab/Qwen3-8B-DFlash-b16` as the draft head. Each cell reports steady-state latency after 10 warmup steps and 50 measurement steps, measured with CUDA events using the PyTorch SDPA backend. For context length L and draft depth N, let $T_{\text{draft}}(N, L)$ denote the latency of one parallel draft-head forward pass that proposes N tokens, and let $T_{\text{verify}}(N, L)$ denote the latency of one parallel target-model verification pass over the same N candidates. Following Eq. equation 1, we define the per-draft-token cost coefficient as

$$c(N, L) = \frac{T_{\text{draft}}(N, L)/N}{T_{\text{verify}}(N, L)} = \frac{T_{\text{draft}}(N, L)}{N\, T_{\text{verify}}(N, L)}.$$

The factor $1/N$ amortizes the parallel draft-head forward cost across the N proposed tokens. Lower c indicates cheaper marginal drafting relative to one target verification pass.

Results. Table 12 reports c in percent over a sweep of context lengths $L \in \{128, \ldots, 4096\}$ and draft depths $N \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512\}$.

Table 12: Per-draft-token drafting cost ratio $c = T_{\text{draft}}/(N\, T_{\text{verify}})$ (%) on a single H200 NVL GPU, sweeping context length L and draft depth N. This is the cost coefficient used in Eq. equation 1 and Fig. 2. Lower is better.

$L \backslash N$	1	2	4	8	16	32	64	128	256	512
128	15.098	6.810	3.396	1.683	0.846	0.422	0.211	0.106	0.053	0.034
256	15.042	6.735	3.329	1.663	0.837	0.419	0.210	0.105	0.052	0.034
512	15.083	6.710	3.327	1.679	0.843	0.421	0.208	0.105	0.052	0.034
1024	15.029	6.720	3.327	1.690	0.845	0.420	0.209	0.104	0.054	0.035
2048	15.031	6.664	3.347	1.668	0.831	0.415	0.211	0.110	0.064	0.036
4096	18.295	8.796	4.411	2.233	1.129	0.566	0.289	0.146	0.073	0.037

Across the practically relevant regime ($L \leq 2048$, $N \geq 16$), the per-draft-token cost is below 1% of one target verification pass and decreases roughly as $1/N$, since the parallel draft-head cost is

amortized across more proposed tokens. At $L = 1024$, for instance, c drops from 0.845% at $N = 16$ to 0.054% at $N = 256$, close to the ultra-low-cost regime in Fig. 2. Longer contexts increase the cost mildly, but c remains below 0.1% for $N \geq 256$.

Implication. These measurements support the main scaling argument: increasing the draft depth N amortizes the draft-head cost and lowers the effective per-token cost coefficient c. On the same hardware, c decreases from $\approx 6.7\%$ at $N = 2$ to $\approx 0.05\%$ at $N = 256$, moving from the typical-cost regime toward the ultra-low-cost regime in Fig. 2. However, as Fig. 2 illustrates, low cost enables long-draft scaling only when acceptance remains sufficiently high. This motivates causal tree drafting as a way to improve acceptance quality in the low-cost speculative decoding.