Dynamic Delayed Tree Expansion For Improved Multi-Path Speculative Decoding

Rahul Thomas[1,2,†,*] **Teo Kitanovski**[1,3,†] **Micah Goldblum**[1,2] **Arka Pal**[1]

[1]Ritual [2]Columbia University [3]Vanderbilt University

Abstract

Multi-path speculative decoding accelerates lossless sampling from a target model by using a cheaper draft model to generate a draft tree of tokens, and then applies a verification algorithm that accepts a subset of these. While prior work has proposed various verification algorithms for i.i.d rollouts, their relative performance under matched settings remains unclear. In this work, we firstly present a systematic evaluation of verification strategies across model families, tasks, and sampling regimes, and find that Traversal Verification dominates consistently, with OT-based methods lagging far behind. Our analysis uncovers that this occurs because OT-based methods achieve high multi-token acceptance near the root of the draft tree, while multi-token gains are most impactful deeper in the draft tree, where draft and target distributions diverge. Based on this insight, we propose **delayed tree expansion**, which drafts a partial single path, delaying the i.i.d. branching point. We show that delayed tree expansion preserves the target distribution and improves on root-node i.i.d rollouts. Further, we develop a dynamic neural selector that estimates the expected block efficiency of OT-based verification methods from draft and target features, enabling context-dependent expansion decisions. Our neural selector allows OT-based methods like SpecInfer to outperform Traversal Verification for the first time, with 5% higher average throughput across a wide range of models, datasets, and sampling settings. Finally, we extend our approach to Traversal Verification to improve its average throughput by 26%.

1 Introduction

In recent years, LLMs have demonstrated impressive performance across a wide range of domains, including but not limited to translation, mathematics, and coding (Zhu et al., 2024; Kasneci et al., 2023; Hendrycks et al., 2021; Chen et al., 2021). Despite such progress, end-to-end latency is a significant bottleneck in real deployments. Most LLM families, such as Gemma (Team, 2024; Team et al., 2025a), Qwen (Bai et al., 2023; Hui et al., 2024), and Llama (Touvron et al., 2023; Grattafiori et al., 2024), are autoregressive and require an expensive forward pass for each output token.

Exploiting the fact that the forward pass is memory-bound (Fu et al., 2024), speculative decoding (Chen et al., 2023; Leviathan et al., 2023) improves GPU utilization by decoding multiple tokens per pass. First, in **drafting**, a cheap draft model proposes multiple tokens. Then, the original LLM performs a **target** forward pass over these tokens in parallel. Finally, during **verification**, some tokens are rejected to maintain the target distribution. Throughput gains depend on the draft and target latencies, as well as block efficiency – the average number of accepted tokens per target pass.

Recent work has generalized speculative decoding to the **multi-path** setting, where the draft model proposes a tree of tokens and the target pass uses a tree attention mask. Multi-path methods improve

[†]Equal contribution.

[*]Corresponding author: rkt2129@columbia.edu.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

block efficiency by diversifying the draft tree. To maintain the target distribution, we require valid multi-path verification algorithms. These are usually based on optimal transport (OT) solvers that traverse the draft tree in a top-down manner (Sun et al., 2023; Khisti et al., 2025; Miao et al., 2024), although some use bottom-up traversal (Sun et al., 2024c; Weng et al., 2025). Concurrently, recent works have improved **drafting** for a fixed verification method via confidence-guided tree construction, adaptive drafting depth, and pipelining (Li et al., 2024b; Brown et al., 2024; Xiong et al., 2025; Chen et al., 2024a; Wang et al., 2025; Huang et al., 2024; Liu et al., 2024b; Guan et al., 2025). These can often only be used with specific verification algorithms, in order to maintain the target distribution.

However, despite the significant body of prior work on multi-path speculative decoding, there has been no systematic and controlled comparison of multi-path verification methods. Therefore, an important open question remains: **Which multi-path verification method performs best, and under what conditions?** Prior works often compare such methods under different drafting methods, datasets, and model families, or only benchmark the most simple baselines and verification methods, making it difficult to measure the true progress of multi-path verification improvements.

In this work, we compare multi-path verification methods under matched i.i.d. draft settings across diverse models, tasks, and sampling regimes. We find that Traversal Verification dominates all other methods. Our analysis reveals that this is the result of wasteful expansion in OT-based methods: they achieve high token acceptances near the draft tree root, but the most promising improvements occur deeper in the tree when draft and target distributions diverge. Generating a draft tree from i.i.d. root rollouts forces token diversity into shallow nodes, even when it is more beneficial at deeper nodes.

This motivates **delayed expansion**, where we draft a partial path and expand i.i.d. rollouts at a branching point. This allows explicit control of three parameters: the single path depth, the expansion factor, and the expansion depth. Optimizing these involves a tradeoff between block efficiency and tree size, because larger trees accept more tokens on average but also incur higher draft and target latency. Following adaptive drafting methods that maximize expected block efficiency offline (Chen et al., 2024a; Brown et al., 2024), we train a lightweight MLP to predict our optimal policy parameters from root node features and draft and target pass times. With this neural selector, OT-based methods finally outperform Traversal Verification. Furthermore, our neural selector can also be applied to Traversal Verification to significantly improve throughput. In summary, our contributions are:

- We provide the first systematic comparison of existing i.i.d. multi-path speculative decoding verification methods across varying model families, datasets, and sampling settings. We find Traversal Verification performs far better than all other methods (OT-based).
- To explain the pitfalls of OT-based methods, we analyze acceptance rates and find that the marginal performance improvements are caused by increasingly diverging target and draft distributions deeper in the tree. Branching early in the tree, where target and draft distributions are most similar, yields relatively less useful acceptance improvements.
- To mitigate this, we utilize delayed expansion, where the draft tree is formed by drafting a path and then branching into many i.i.d. paths. We train a context-dependent neural predictor on offline block efficiency estimates to predict optimal path and branch parameters from model and latency features. In online deployments, our method can be applied to all OT-based i.i.d. multi-path methods as well as Traversal Verification.
- Across a multitude of settings, our neural predictor with the OT-based method SpecInfer improves average throughput over Traversal Verification by 5%, and our neural predictor with Traversal Verification improves average throughput on Traversal by 26%.

2 Background

Speculative decoding uses a draft model to decode many tokens in one target model call:

1. **Drafting.** Using only the cheap draft model, we expand a tree of tokens from the current context and compute next-token draft distributions on the tree nodes.
2. **Target Pass.** The target model does a forward pass over the draft tree with a custom attention mask for ancestor-only dependencies, to get next-token target distributions on nodes.
3. **Verification.** Using target and draft distributions, we randomly accept one node on the tree and append one correction token, such that the output follows the target model distribution.

The **acceptance length** τ is the depth of the accepted node, which grows as the target and draft distributions coincide. The **block efficiency** $\mathbb{E}[\tau + 1]$ is the average number of decoded tokens per target call. When drafting is cheap and the batched target pass takes as long as a target forward pass, block efficiency accurately represents speedup.

2.1 Performance Comparisons

The literature lacks comprehensive comparisons of verification algorithms. Xia et al. (2024) surveys existing drafting and verification approaches and releases Spec-Bench, a third-party testing environment for speculative decoding. However, their dataset is fairly small, they only support a few verification algorithms, and their evaluations do not use sampling variations. More recently, Liu et al. (2025) gives the first production-grade vLLM study (Kwon et al., 2023) on the effects of batch size, model family, and workloads on speculative decoding and variants like EAGLE-3 and multi-token prediction, but they use only basic verification methods and greedy decoding. We are the first to systematically compare i.i.d. multi-path verifiers across models, tasks, and sampling settings.

2.2 Improving Drafting

Existing work has focused on improving block efficiency by modifying drafting or verification. Verification works build novel algorithms that alter the node selection scheme to improve acceptance lengths. Note that verification and draft improvements can be integrated together when compatible, and many of these drafting methods rely on specific verification methods like NSS (Miao et al., 2024), Naive (Chen et al., 2023; Leviathan et al., 2023), and SpecInfer (Miao et al., 2024). We discuss these in Section 3 and focus on drafting here. We focus on works that alter the draft tree construction itself, as these are most relevant to our work. We provide further details on orthogonal work in Section A.

Context-dependent tree structures. Various works alter the draft topology *online* to only expand nodes with meaningful acceptance gains. EAGLE-2 (Li et al., 2024b) deterministically expands top-k tokens from nodes with the highest global acceptances, using draft confidence as a proxy for acceptance. Dynamic Depth Decoding (Brown et al., 2024) improves EAGLE-2 by adaptively selecting the expansion depth in the deterministic tree. EDD and PCT (Zheng and Wang, 2025) prune branches with low draft confidence, so that the target model compute is spent on branches likely to be accepted. Instead of fixing a deterministic tree, DySpec (Xiong et al., 2025) dynamically allocates a base tree expansion budget and then samples a draft tree from the base tree online. Our methods also employ context-dependent tree topologies, but they can more effectively push towards high acceptance regions by relying on offline block efficiency supervision rather than simple proxies.

Offline tree optimization for block efficiency. Another line of work uses explicit objectives for expected accepted length to select a set of optimal trees offline. Sequoia (Chen et al., 2024a) uses dynamic programming to select deterministic draft structures under various budget constraints, which optimize SpecInfer block efficiency for sampling without replacement. OPT-Tree (Wang et al., 2025) instead maximizes expected acceptance under NSS. While these methods are similar to ours in that they optimize expected block efficiency, their fixed offline tree collection remains highly sensitive to the dataset and sampling settings, a pitfall which our per-context neural selector naturally overcomes.

Dynamic draft length control. The search over tree topologies is complex, so many draft tree selectors decide how far to draft rather than changing branching logic. SpecDec++ (Huang et al., 2024) learns an adaptive stopping policy to decide when draft tokens bring minimal benefits. SVIP (Zhang et al., 2024) uses a draft confidence threshold to terminate early when rejections are likely. FailFast (Pan et al., 2025) extends this by using dLLMs for rapid draft generation, and rapidly increasing draft length in high-acceptance regions to accept many tokens. AdaSD (Lu et al., 2025) adapts draft length by using target and draft model divergence as a proxy for acceptance.

Training-based tree policy. Draft model language modeling performance is not highly correlated with speculative decoding performance (Yan et al., 2025). Thus, some works train draft models on objectives aligned with speculative decoding. DistillSpec (Zhou et al., 2023) uses knowledge distillation to align the draft model to the target model for Naive acceptance. Online speculative decoding (Liu et al., 2023) uses knowledge distillation to update the draft model during inference, dynamically clustering queries by common domain or topic to improve alignment. AdaSpec (Hu et al.,

2025b) adapts speculative decoding to latency constraints in model serving, prioritizing response time over block efficiency. Group Tree Optimization (Hu et al., 2025a) uses an NSS-based tree reward and a PPO-style draft tree objective to better align distillation with inference-time drafting. Our work differs from these as it expands NSS and Naive training objectives to more diverse verification methods, and it trains a cheap tree selector model instead of finetuning a large draft model.

Hardware-aware tree decoding. Some works accelerate the batched forward pass or pick tree shapes that are highly compatible with compilers. Sequoia (Chen et al., 2024a) uses hardware-dependent target and draft times to inform its tree depth and branch selections. Yggdrasil (Guan et al., 2025) uses an equal-growth drafting algorithm to select tree width and depth and verification width to optimize the graph compiler, and improves speculative decoding stage scheduling for GPU to CPU transfer. DeFT (Yao et al., 2024) uses QKV grouping and custom kernels to accelerate tree decoding, which is highly relevant to the drafting phase. While we train our neural selector using draft and target pass times, and are hardware-aware like these works, our approach differs by using verifier-specific block efficiency estimates to optimize delayed i.i.d. trees.

3 Verification Algorithms

We now review prior verification algorithms, shown in Table 1. The target and draft models are M_p, M_q and share a vocabulary $\mathcal{V}$. We let $p(\cdot|\boldsymbol{c}), q(\cdot|\boldsymbol{c})$ be their next-token distributions on $\mathcal{V}$. All algorithms require a draft tree, which we formalize below. Here, $(\boldsymbol{c}_1, \boldsymbol{c}_2)$ is the concatenation of contexts. We ignore the root context $\boldsymbol{c}$ in our p, q and node notation when it is implicitly clear.

Definition 3.1. *A **draft tree** rooted at $\boldsymbol{c}$ is a directed tree of nodes represented by distinct contexts $\boldsymbol{c}'$ with root node $\boldsymbol{c}$, such that for each parent and child node pair $(\boldsymbol{c}_p, \boldsymbol{c}_c)$, we have $\boldsymbol{c}_c = (\boldsymbol{c}_p, t)$ for some token $t \in \mathcal{V}$. We denote the list of child nodes by $ch(\boldsymbol{c})$, which can duplicate nodes*[1].

For a verification algorithm to preserve the target distribution, it must be compatible with drafting. We cover those which apply to single-path, multi-path, and other drafting regimes. Single-path means the tree is a path sampled from M_q, and multi-path is the union of i.i.d. single-paths.

Table 1: We consider the following single-path, multi-path, and OT-based verification algorithms.

Algorithm	Appears In	Multi-Path	OT-Based
NaiveTree	Leviathan et al. (2023)	✓	✓
NSS	Miao et al. (2024)	✓	✓
SpecTr	Sun et al. (2023)	✓	✓
SpecInfer	Miao et al. (2024)	✓	✓
Khisti	Khisti et al. (2025)	✓	✓
Naive	Chen et al. (2023)		
TV	Hu and Huang (2024)		
BV	Sun et al. (2024c)		
Traversal	Weng et al. (2025)	✓	

3.1 Single-Path Algorithms

To the best of our knowledge, there are only three unique[2] verification algorithms that apply when the draft tree is an autoregressively sampled length L block $a_{1:L} \sim q(\cdot|\boldsymbol{c})$. Furthermore, unlike the multi-path setting, the optimal algorithm in this setting is already known: block verification.

Naive. Speculative sampling Chen et al. (2023); Leviathan et al. (2023) independently accepts each node $a_{1:i}$ with probability $\min(1, p(a_i|\boldsymbol{c}, a_{1:i-1})/q(a_i|\boldsymbol{c}, a_{1:i-1}))$. Then, it selects the maximal depth node $a_{1:\tau}$ with *all* ancestor nodes accepted. Finally, it samples a correction token from a residual distribution proportional to $\max\{p(\cdot|a_{1:\tau}) - q(\cdot|a_{1:\tau}), 0\}$ if $\tau < L$ and $p(\cdot|a_{1:\tau})$ otherwise.

[1] This is not standard definition of a child node list in a tree, as child nodes have multiplicity and their order matters.

[2] All multi-path algorithms apply here, but all OT-based ones except NSS degenerate to Naive, and Traversal degenerates to BV. NSS degenerates to a simple algorithm: sample from p until the trajectory no longer matches the draft block.

Tree verification. Tree verification (Hu and Huang, 2024) provably improves naive speculative sampling with Monte Carlo tree sampling. We do not use it because block verification is faster.

Block verification (BV). Block verification (Sun et al., 2024c) provably improves upon naive speculative sampling by relaxing the requirement that all ancestors must be accepted. They recursively define node weights $w(a_{1:i}) = \min(1, w(a_{1:i-1})p(a_i|a_{1:i-1})/q(a_i|a_{1:i-1}))$ and independently accept each node $a_{1:i}$ according to these weights. They return the maximal depth node $a_{1:\tau}$ and sample the correction token from a w-weighted naive residual. As Sun et al. (2024c) show in their paper, block verification achieves the highest block efficiency among any single-path verification algorithm that only takes in p and q data on $a_{1:L}$, so it is also provably better than tree verification.

3.2 Multi-Path Algorithms

Now, we consider draft trees which are the union of K i.i.d. length L paths $a[k]_{1:L} \sim q_L(\cdot|\boldsymbol{c})$. When paths overlap, for every parent-child node pair $(\boldsymbol{c}_p, \boldsymbol{c}_c)$, the multiplicity of $\boldsymbol{c}_c$ in the child node list of $\boldsymbol{c}_p$ is the number of times $\boldsymbol{c}_c$ appears as a prefix of some $a[k]$. This construction is practical for speculative decoding due to GPU parallelization. For moderate K, L, there is little overhead in the *batched* forward pass across K sequences, relative to a single forward pass (Agrawal et al., 2024a; Dao et al., 2022). The majority of multi-path algorithms here are OT-based and use OTLP solvers.

Definition 3.2. *An **OTLP solver** $f_{p,q,k} : \mathcal{V}^k \to \mathcal{V}$ on target and distributions $p, q \in \Delta(\mathcal{V})$ with multiplicity $k \in \mathbb{N}$ is a randomized function where $f(X_1, \ldots, X_k) \sim p$ for i.i.d. $X_1, \ldots, X_k \sim q$.*

OTLP solvers perform **top-down** draft tree traversal. At each node $\boldsymbol{c}$, taking in $p(\cdot|\boldsymbol{c}), q(\cdot|\boldsymbol{c})$ and the child nodes $\text{ch}(\boldsymbol{c}) = [(\boldsymbol{c}, x_i)]_{i=1}^k$, an OT-based solver computes a feasible solution of an optimal transport linear program to append a token t to $\boldsymbol{c}$, progressing to the child node $(\boldsymbol{c}, t)$ if $t \in \{x_1, \ldots, x_k\}$ and stopping otherwise. The OT-based algorithms we consider in the i.i.d. multi-path setting are NSS (Miao et al., 2024), SpecInfer (Miao et al., 2024), SpecTr (Sun et al., 2023), and Khisti (Khisti et al., 2025). Also, while speculative sampling (Leviathan et al., 2023; Chen et al., 2023) was presented as a single-path algorithm, it is OT-based and can also be made multi-path: to distinguish this from Naive, we call it NaiveTree. We review these details in Section B. Rowan et al. (2026) also propose a top-down multi-draft verification method using Gumbel-based list-level coupling. Although their method does not explicitly solve an optimal transport linear program, it still fits our definition of an OT-based verifier and can be integrated into our framework. We leave the details to future work.

Traversal verification. Weng et al. (2025) gives the only multi-path algorithm that traverses the draft tree from the **bottom-up**. When $K = 1$, this reduces to block verification.

We stress that these algorithms may outperform each other in different (p, q) regimes, which depend on the model, dataset, and p-sampling setting. On the OT-based side, recent work (Hu et al., 2025c; Thomas and Pal, 2025) has computed the acceptance rate of an optimal or near-optimal OTLP solver in certain cases, but a fully optimal OT-based solver remains infeasible. To the best of our knowledge, there are no existing comparisons between Traversal Verification and *all* OT-based methods.

3.3 Other Algorithms

For deterministic trees, used in works like EAGLE (Li et al., 2024a), only NSS (Miao et al., 2024) can be used. Some works also mix i.i.d. and deterministic constructions. For example, Hu et al. (2025c) devise an OT-based algorithm where the top-$(k-1)$ draft tokens are chosen deterministically and the last is random without replacement, with SpecHub (Sun et al., 2024b) being equivalent to the case $k = 2$. We leave exploration of these methods in the context of dynamic drafting to future work.

4 Comparison of Existing Verification Algorithms

In this section, we benchmark the verification algorithms from Section 3: two single-path algorithms (Naive, BV); and 6 multi-path algorithms (NSS, NaiveTree, SpecTr, SpecInfer, Khisti, Traversal). We evaluate the target-draft model pairs Llama-3 70B/8B-Instruct (Grattafiori et al., 2024), Gemma-3 27B/270M-IT (Team et al., 2025b), and Qwen-2.5 32B/0.5B-Instruct (Qwen et al., 2025). Our selection spans varying model families, target model sizes, and target-to-draft size ratios. We focus

Table 2: Average block efficiency (tokens/call) and throughput (tokens/sec) across datasets and sampling configurations of existing verification algorithms. For more detailed data, see Section H.

Method	**Block Efficiency** (tokens/call)				**Throughput** (tokens/sec)			
	Qwen	**Gemma**	**Llama**	**Average**	**Qwen**	**Gemma**	**Llama**	**Average**
NaiveTree	5.00	2.01	6.50	4.50	19.16	6.00	14.71	13.29
NSS	4.44	1.99	5.72	4.05	16.88	5.94	12.91	11.91
SpecTr	5.11	2.05	6.67	4.61	19.01	6.09	**14.86**	13.32
SpecInfer	5.13	2.07	6.55	4.58	19.58	6.21	**14.86**	13.55
Khisti	4.90	2.05	6.29	4.41	18.67	5.93	13.66	12.75
Naive	4.60	3.44	5.52	4.52	16.55	9.20	11.58	12.44
BV	4.24	3.30	5.37	4.30	17.30	10.57	12.20	13.36
Traversal	**5.33**	**3.81**	**6.78**	**5.31**	**21.35**	**11.56**	14.77	**15.89**

on two metrics: **block efficiency** and **throughput**. Block efficiency improves as the draft tree grows, but speedups tend to follow a U-curve, as larger trees incur increased latency in the target and draft forward passes. Thus, throughput is more relevant in deployment. Still, block efficiency is useful to report as unlike throughput, it is not dependent on systems parameters and attention kernels.

4.1 Experimental Setup

We select 50 prompts from each of 5 datasets: MATH500 (Lightman et al., 2023), OlympiadBench (He et al., 2024), LiveCodeBench (Jain et al., 2024), LitBench (Fein et al., 2025), and Opus (Zhang et al., 2020; Tiedemann, 2012). In LiveCodeBench, we use the 'code_generation_lite' subset; for Opus, we select half of our prompts each from 'opus_books' and 'open_subtitles' with a uniform split of translations between English, French, Spanish, and Italian. These datasets represent diverse generative settings: MATH500 and OlympiadBench test math reasoning with distinct difficulties, LiveCodeBench tests coding, LitBench tests creative writing, and Opus tests translation. We use 8 batch size 1 sampling configurations: sampling from M_p with temperatures $\{0.2, 0.4, 0.6, 0.8, 1.0, 1.2\}$ and no nucleus sampling, and sampling from M_p with temperature 1.0 and nucleus sampling in $\{0.9, 0.99\}$. We use 2 A100-80GB GPUs with an Intel Xeon Gold 6342 CPU (12 cores, 2.80 GHz).

4.2 Results

We compare existing verification algorithms in Table 2, averaged across all datasets and sampling configurations per model. For each sampling configuration and dataset, we select parameters $K \leq 4, L \leq 8$ that maximize block-efficiency or throughput. Detailed breakdowns are in Section H.

We see that Traversal wins in both block efficiency and throughput, beating the second-best method in each by $\sim$15%. Traversal is best on Gemma, but is also the top method for Qwen and only slightly below the best Llama throughput. Analysis in Section H shows that Traversal is consistently the best in throughput and block efficiency across all configurations. We hypothesize this is because the top-down approach in OT-based methods makes deep traversal in tree exponentially difficult, whereas the bottom-up approach in Traversal from leaf nodes is more likely to accept longer sequences.

After Traversal, we find that SpecTr, BV, and SpecInfer perform similarly in average throughput. NSS performs worst, which is expected as it does not use draft probabilities in verification. Interestingly, OT-based algorithms perform different in the three families: in throughput, SpecInfer wins on Qwen, Naive wins on Gemma, and SpecTr and SpecInfer tie on Llama, with Khisti not lagging far behind.

5 Delayed Tree Expansion

While our results in Section 4 show that Traversal Verification outperforms OT-based methods in all settings, we also see there is no dominant winner among OT-based methods. To explain this, we examine acceptance rates of OT-based methods at tree nodes, which measure how often an OT-based method appends a token that stays on the draft tree. These do not depend on the observed child nodes, rather only on p and q. We show how to compute these rates for OT-based methods in Section C.

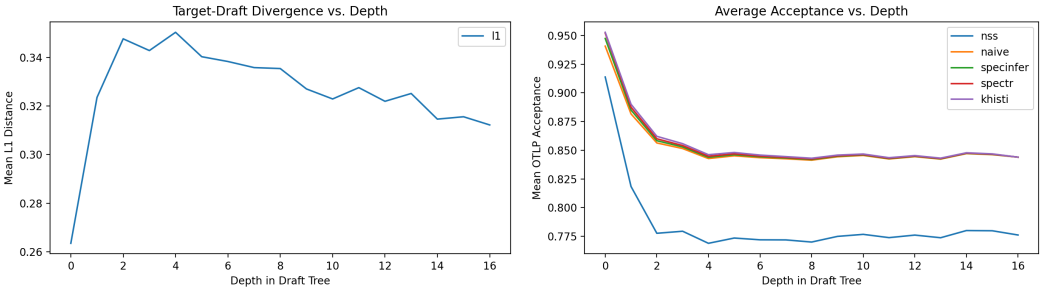

Figure 1: We generate 200,000+ draft (Llama-3 8B-Instruct) trees from roots of target model (Llama-3 70B-Instruct) trajectories and compute both L_1 target-draft distance and average OTLP acceptances across varying draft tree depths. The divergence between target and draft distributions spikes deeper in the tree, and acceptances across all OTLP methods consistently decrease with depth.

Definition 5.1. *For any OTLP solver $f_{p,q,k}$, its **acceptance rate** $\alpha(f_{p,q,k})$ is the probability that its output token lies among its input tokens, for prior i.i.d. input samples $X_1, \ldots, X_k \sim q$.*

We follow Section 4, but now analyze *offline* trees generated from over 200,000 draft roots along fixed target model trajectories, taken at every 16 tokens. We use vLLM (Kwon et al., 2023) to speed up data generation by parallelizing across roots in a prompt. Our aggregate Llama results for all 8 sampling settings on LiveCodeBench, MATH500, and OlympiadBench are shown in Figure 1, alongside target-draft L_1 distances at the same nodes.

We observe for all OT-based methods that node-wise acceptance rates rapidly degrade with depth. This is explained by the spike in target-draft L_1 distance near depth 2. Even though L_1 decreases at later depths, it is still 20% higher than the root L_1 at depth 16. When $\|p - q\|_1$ is large deep in the tree, acceptances degrade: for example, at nodes with one child node, all OT-based methods reduce to Naive, whose acceptance reduces linearly with L_1 distance. While we often branch early in the tree and improve acceptance near the root, it can be better to improve acceptances deeper in the tree.

Motivated by these findings, we define a delayed tree, which drafts a partial path and then expands. OTLP solvers can still be used on such a tree while preserving the target distribution, because conditioned on their current node, their output token follows the target distribution p (Section D).

Definition 5.2. *Fix $K, L_1, L_2 \in \mathbb{N}$ with $K \geq 1$. A **random** (K, L_1, L_2)**-delayed-tree** at context $\boldsymbol{c}$ samples a length-L_1 path $a_{1:L_1} \sim q(\cdot|\boldsymbol{c})$, and branches into K i.i.d. length-L_2 paths $a[k]_{1:L_2} \sim q(\cdot|\boldsymbol{c}, a_{1:L_1})$. Child node lists have duplicates if paths overlap after the branching point $(\boldsymbol{c}, a_{1:L_1})$.*

Now that we have motivated delayed expansion, we discuss its evaluation. How do we estimate the block efficiency of an OT-based method on a delayed tree, when both drafting and verification involve randomness? Motivated by adjacent work in adaptive drafting (Chen et al., 2024a; Brown et al., 2024), we define the notion of branching probability for an OTLP solver.

Definition 5.3. *For any OTLP solver $f_{p,q,k}$ with multiplicity $k \in \mathbb{N}$, list of k tokens $\boldsymbol{x}$, and token $t \in \mathcal{V}$, the **branching probability** to t, $B(f_{p,q,k}, \boldsymbol{x}, t)$, is the probability that $f_{p,q,k}(\boldsymbol{x}) = t$.*

This can be used to compute expected block efficiency, using the Law of Total Expectation to separate randomness over draft tree (K, L_1, L_2)-delayed-trees $\mathcal{T}$ from verification algorithm randomness:

$$\mathbb{E}[\tau + 1] = \mathbb{E}_{\mathcal{T}}\left[\mathbb{E}[\tau + 1|\mathcal{T}]\right] = \mathbb{E}_{\mathcal{T}}\left[\sum_{\boldsymbol{c}' \in \mathcal{T}} \mathbb{P}(\text{OTLP solver reaches } \boldsymbol{c}'|\mathcal{T})\right]. \tag{1}$$

The last equality is from the definition of an OT-based algorithm in Section 3.2: in verification on tree $\mathcal{T}$, a node $\boldsymbol{c}'$ adds to $\tau + 1$ if and only if the OTLP solver reaches $\boldsymbol{c}'$. Now, because OT-based methods iteratively append the output of $f_{p,q,k}$ to progress down the draft tree, we simplify this to

$$\mathbb{E}[\tau + 1] = \mathbb{E}_{\mathcal{T}}\left[\sum_{(\boldsymbol{c}, t_1, \ldots, t_d) \in \mathcal{T}} \prod_{j=1}^{d} B\left(f_{p,q,k}, \text{ch}(\boldsymbol{c}, t_1, \ldots, t_{j-1}), t_j\right)\right], \tag{2}$$

where p, q in $f_{p,q,k}$ implicitly represent node distributions $p(\cdot|\boldsymbol{c}, t_1, \ldots, t_{j-1}), q(\cdot|\boldsymbol{c}, t_1, \ldots, t_{j-1})$. We estimate this by taking an average of the inner sum over $s = 4$ delayed tree samples $\mathcal{T}_1, \ldots, \mathcal{T}_s$. This does not eliminate drafting variance, but eliminates verification variance and is unbiased.

Traversal Verification. Traversal Verification is not OT-based, so Equation (2) cannot be used. Nevertheless, in Section F, we derive an efficient algorithm to estimate $\mathbb{E}[\tau + 1] = \mathbb{E}_{\mathcal{T}}[\tau + 1|\mathcal{T}]$ for Traversal. Our approach relies on the fact that the leaf-removal order in Traversal is deterministic. Traversal also preserves the target distribution p on delayed trees (Section D).

6 Neural Delayed Expansion

Delayed tree expansion from Section 5 introduces discrete design choices that influence end-to-end throughput. For a fixed verifier, the throughput of a delayed expansion policy is a tradeoff between accepted tokens and drafting and target pass latency. As p, q vary significantly across contexts, a static (K, L_1, L_2) configuration is not always optimal across requests. Thus, we devise a context-dependent neural selector that predicts which K, L_1, L_2 values maximize throughput at each decoding step.

Neural delayed expansion (NDE) generalizes delayed expansion. At context $\boldsymbol{c}$, our neural selector outputs a policy $\pi_\theta(\cdot|\boldsymbol{c})$ over actions $a = (K, L_1, L_2) \in \mathcal{A} = \{1, 2, 3, 4\} \times \{0, \ldots, 8\}^2$ for drafting. The selector must choose a before drafting, so we only use features that are efficient to obtain at the root, including draft and target hidden states at the previous token and scalar features on sampling, entropy, divergence, and latency. We design our selector to project hidden states to a shared dimension, apply normalization, concatenate scalar features, and apply a two-layer MLP to produce logits over supported actions. Given the logits, we can then define the offline throughput estimate at context $\boldsymbol{c}$ as the ratio of expected block efficiency to expected wall-time under the categorical policy $\pi_\theta(\cdot|c)$:

$$\text{TPS}_{\pi_\theta}(\boldsymbol{c}) = \frac{\mathbb{E}_{a\sim\pi_\theta(\cdot|\boldsymbol{c})}[\tau + 1|\boldsymbol{c}, a]}{\mathbb{E}_{a\sim\pi_\theta(\cdot|\boldsymbol{c})}[\mathcal{W}(\boldsymbol{c}, a)]}. \tag{3}$$

For minibatch size B, we define our minimization objective as below in Equation (13). We include a penalty term for regressing below the baseline, where λ is the penalty weight and $\mathcal{I}_\alpha$ are the top-α-fraction indices of penalty terms $\max\{1 - \text{TPS}_\pi(\boldsymbol{c}_i)/\text{TPS}_{\text{base}}(\boldsymbol{c}_i), 0\}$ in the minibatch $\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_B\}$:

$$\mathcal{L} = \frac{1}{B}\sum_{i=1}^{B}\left(-\log\frac{\text{TPS}_\pi(\boldsymbol{c}_i)}{\text{TPS}_{\text{base}}(\boldsymbol{c}_i)}\right) + \frac{\lambda}{|\mathcal{I}_\alpha|}\sum_{i\in\mathcal{I}_\alpha}\max\left(1 - \frac{\text{TPS}_\pi(\boldsymbol{c}_i)}{\text{TPS}_{\text{base}}(\boldsymbol{c}_i)}, 0\right)^2. \tag{4}$$

Here, TPS_{base} is throughput for the static baseline. At inference time, the selector produces $\pi_\theta(\cdot|\boldsymbol{c})$ once per decoding step. We then select the action a^* with the highest assigned probability $\pi_\theta(a^*|\boldsymbol{c})$, draft an a^*-delayed tree, and proceed with the target forward pass and verification. Due to space constraints, we provide full details on feature and loss design, as well as estimation of the TPS numerator and denominator expectations, in Section G, with the full algorithm in Algorithm 17.

We formally prove that NDE is lossless in Section D. Since the neural selector chooses the delayed expansion policy before drafting, the distribution preservation guarantees for fixed delayed trees also extend to these adaptive delayed trees, for both OT-based verifiers and Traversal Verification.

7 Experiments

We now evaluate our neural selector (NDE) following the experimental setup in Section 4.1. We train on offline data from 350 prompts per dataset, and measure performance on a held-out test set.

First, we compare NDE to baseline algorithms in Table 3. We compute improvement ratios in block efficiency and TPS per verification method. Prompt-level SEs for TPS ratios are in Table 18, including improved Llama results (Section 7.1). We see that NDE improves on the baseline algorithms by around 10% and 25% in block efficiency and throughput on average across models. Strong gains are obtained on Gemma, which is up to $2.2\times$ faster than the baseline. We speculate this is because Gemma has the highest target-to-draft size ratio. This induces divergent proposal distributions that static baselines cannot overcome, but NDE can with dynamic selection. To understand the relative importance of input NDE features in these improvements, we perform ablations in Section J. NDE is robust to changes in dropout and noise, and can even perform well with only scalar features.

Table 3: Block efficiency and throughput ratio improvements for NDE over the corresponding baseline algorithms, averaged across datasets and sampling configurations. See Section 7.4 for Traversal NDE.

Method	Block Efficiency Ratio				Throughput Ratio			
	Qwen	Gemma	Llama	Average	Qwen	Gemma	Llama	Average
Khisti NDE	1.01	1.90	1.09	1.19	0.99	2.17	0.97	1.16
NaiveTree NDE	1.07	0.99	1.17	1.09	1.31	1.40	1.21	1.30
NSS NDE	0.97	1.68	1.01	1.11	1.20	2.09	0.96	1.26
SpecInfer NDE	0.98	1.65	1.01	1.10	1.15	2.14	0.96	1.23
SpecTr NDE	0.97	1.68	1.01	1.10	1.14	2.17	0.96	1.23

Table 4: Average block efficiency and throughput across datasets and sampling settings. Traversal is the best existing algorithm; NDE is our approach. Full results are in Section H.

Method	**Block Efficiency** (tokens/call)				**Throughput** (tokens/sec)			
	Qwen	Gemma	Llama	Average	Qwen	Gemma	Llama	Average
Traversal	**5.33**	3.81	6.78	**5.31**	21.35	11.56	**14.77**	15.89
Khisti NDE	4.97	**3.89**	**6.84**	5.23	18.41	12.84	13.24	14.83
NSS NDE	4.31	3.34	5.78	4.48	20.29	12.40	12.45	15.05
NaiveTree NDE	4.94	3.42	6.47	4.94	21.67	12.86	14.01	16.18
SpecTr NDE	4.97	3.44	6.74	5.05	21.74	13.22	14.23	16.40
SpecInfer NDE	5.03	3.42	6.64	5.03	**22.54**	**13.26**	14.27	**16.69**

We also compare the best performing verification method, Traversal, to our NDE methods in Table 4. We see that by applying NDE, we are able to eliminate the gap between OT methods and Traversal. Now, SpecInfer NDE is able to outperform Traversal in throughput by around 5% on average.

7.1 Improving Llama

To explain why NDE works better on Gemma than Llama, we analyze how often the ground-truth oracle in our Section 5 experiments sets $L_2 = 8$, the upper bound, to maximize block efficiency. Llama has the highest frequency at 55%, followed by Qwen at 44% and Gemma at 37%. This aligns with NDE improvements on OT-based methods: Gemma is the best, Qwen has significant gains, and Llama is stagnant. For models like Llama, where oracle parameters have little variance, NDE cannot improve the best static baseline significantly. Nevertheless, in Section I, we explore various regularization techniques to prevent overfitting Llama's larger hidden state. With these modifications, we are able to make all NDE methods improve upon Llama baselines, with gains from 2% to 12%.

7.2 OOD Generalization

We test SpecInfer NDE, which was trained on translation, writing, math, and coding data, on three other domains: CRAG for RAG, GovReport for summarization, and IFBench for instruction-following (Yang et al., 2024; Huang et al., 2021; Pyatkin et al., 2025). Block efficiency gains are moderate, ranging from 1.1 to 1.2× over the baseline. However, throughput improvements are strong, as we achieve 2.37×, 2.30×, and 2.79× over the baseline on CRAG, GovReport, and IFBench. This metric discrepancy is likely caused by the SpecInfer baseline wasting compute in low acceptance regions.

7.3 Dynamic Drafting Baselines

We compare Naive NDE to dynamic drafting baselines AdaEDL (Agrawal et al., 2024b) and Sequoia (Chen et al., 2024a). We show how to adapt these methods to delayed expansion in Section K. Naive NDE is faster than both across all settings and doubles Sequoia throughput on coding and math.

7.4 Traversal NDE

Following Section 5 and Section F, we can apply NDE to Traversal. We use our existing Qwen, Gemma, and Llama setups, and also test Olmo-2-32B/1B-Instruct (OLMo et al., 2024). Average throughput improvements across all datasets and sampling configurations are 1.22× for Qwen, 1.71× for Gemma, 1.00× for Llama, and 1.12× for Olmo, with 1.26× average across all settings. These gains are statistically significant; for example, Qwen and OLMo have prompt-level standard errors of 0.019 and 0.018, which are much smaller than their gains. **Traversal NDE beats Traversal and all other NDE methods by over 20%, achieving a new state-of-the-art.**

7.5 Deployment Costs

NDE incurs one-time offline costs from trace generation and block efficiency estimation. For Qwen , these are estimated at 322 GPU hours and 80-160 GPU hours per verifier, respectively. At inference time, NDE requires an extra draft forward pass but no extra target pass. Across Qwen SpecInfer-NDE runs, the full policy overhead is profiled at 0.55% of end-to-end walltime on Blackwell RTX Pro 6000 GPUs. When transferring NDE to new hardware, traces and block efficiency estimates can be reused, requiring only latency recalibration and lightweight selector retraining.

8 Conclusion

We conducted the first systematic study of i.i.d. multi-path verification algorithms. We found that Traversal Verification outperforms all OT-based methods. By analyzing target-draft divergence and acceptance rates, we found this was caused by wasteful branching early in the draft tree. To overcome this, we trained a neural selector to optimally select depth and expansion parameters based on a throughput objective. With this neural selector, OT-based methods beat Traversal, and Traversal significantly improves. Future work could consider larger-scale evaluations across datasets and generation lengths, more robust training objectives for block efficiency, or integration with other dynamic drafting works.

Acknowledgments and Disclosure of Funding

Funding: Ritual funded the Paperspace compute used to conduct the experiments in this work.

Competing interests: All authors were employed by Ritual at the time of this work.

References

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI '24)*, 2024a. URL https://www.usenix.org/system/files/osdi24-agrawal.pdf.

Sudhanshu Agrawal, Wonseok Jeon, and Mingu Lee. Adaedl: Early draft stopping for speculative decoding of large language models via an entropy-based lower bound on token acceptance probability. *arXiv preprint arXiv:2410.18351*, 2024b.

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

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Oscar Brown, Zhengjie Wang, Andrea Do, Nikhil Mathew, and Cheng Yu. Dynamic depth decoding: Faster speculative decoding for llms. *arXiv preprint arXiv:2409.00142*, 2024.

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.

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.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pinto, Jared Kaplan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *arXiv preprint arXiv:2402.12374*, 2024a.

Ziyi Chen, Xiaocong Yang, Jiacheng Lin, Chenkai Sun, Kevin Chang, and Jie Huang. Cascade speculative drafting for even faster llm inference. *Advances in Neural Information Processing Systems*, 37:86226–86242, 2024b.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems (NeurIPS 2022)*, 2022. URL https://openreview.net/pdf?id=H4DqfPSibmx.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

Daniel Fein, Sebastian Russo, Violet Xiang, Kabir Jolly, Rafael Rafailov, and Nick Haber. Litbench: A benchmark and dataset for reliable evaluation of creative writing, 2025. URL https://arxiv.org/abs/2507.00769.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 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.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Yue Guan, Changming Yu, Shihan Fang, Weiming Hu, Zaifeng Pan, Zheng Wang, Zihan Liu, Yangjie Zhou, Yufei Ding, Minyi Guo, et al. Yggdrasil: Bridging dynamic speculation and static runtime for latency-optimal tree-based llm decoding. *arXiv preprint arXiv:2512.23858*, 2025.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems, 2024.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. Rest: Retrieval-based speculative decoding. *arXiv preprint arXiv:2311.08252*, 2023.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. URL https://arxiv.org/abs/1606.08415.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Shijing Hu, Jingyang Li, Zhihui Lu, and Pan Zhou. Bridging draft policy misalignment: Group tree optimization for speculative decoding. *arXiv preprint arXiv:2509.22134*, 2025a.

Yuezhou Hu, Jiaxin Guo, Xinyu Feng, and Tuo Zhao. Adaspec: Selective knowledge distillation for efficient speculative decoders. *arXiv preprint arXiv:2510.19779*, 2025b.

Zhengmian Hu and Heng Huang. Accelerated speculative sampling based on tree monte carlo. In *Forty-first International Conference on Machine Learning*, 2024.

Zhengmian Hu, Tong Zheng, Vignesh Viswanathan, Ziyi Chen, Ryan A Rossi, Yihan Wu, Dinesh Manocha, and Heng Huang. Towards optimal multi-draft speculative decoding. *arXiv preprint arXiv:2502.18779*, 2025c.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. Specdec++: Boosting speculative decoding via adaptive candidate lengths. *arXiv preprint arXiv:2405.19715*, 2024.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In *Proceedings of the 2021 conference of the north American chapter of the association for computational linguistics: Human language technologies*, pages 1419–1436, 2021.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

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.

Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, Stephan Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Juergen Pfeffer, Oleksandra Poquet, Michael Sailer, Albrecht Schmidt, Tina Seidel, and Gjergji Kasneci. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103: 102274, 01 2023. doi: 10.1016/j.lindif.2023.102274.

Ashish J Khisti, MohammadReza Ebrahimi, Hassan Dbouk, Arash Behboodi, Roland Memisevic, and Christos Louizos. Multi-draft speculative sampling: Canonical decomposition and theoretical limits. In *The Thirteenth International Conference on Learning Representations*, 2025.

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

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

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

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.

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.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding for accelerating llms via double early exiting. *Advances in Neural Information Processing Systems*, 37:11946–11965, 2024a.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. Pearl: Parallel speculative decoding with adaptive draft length. *arXiv preprint arXiv:2408.11850*, 2024b.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. *arXiv preprint arXiv:2310.07177*, 2023.

Xiaoxuan Liu, Jiaxiang Yu, Jongseok Park, Ion Stoica, and Alvin Cheung. Speculative decoding: Performance or illusion? *arXiv preprint arXiv:2601.11580*, 2025.

Kuan-Wei Lu, Ding-Yong Hong, and Pangfeng Liu. Adasd: Adaptive speculative decoding for efficient language model inference. *arXiv preprint arXiv:2512.11280*, 2025.

Zhiyao Ma, In Gim, and Lin Zhong. Cacheback: Speculative decoding with nothing but cache. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 31067–31072, 2025.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949, 2024.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*, 2024.

Rui Pan, Zhuofu Chen, and Ravi Netravali. Fail fast, win big: Rethinking the drafting strategy in speculative decoding via diffusion llms. *arXiv preprint arXiv:2512.20573*, 2025.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. *arXiv preprint arXiv:2507.02833*, 2025.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Joseph Rowan, Truong Buu Phan, and Ashish Khisti. List-level distribution coupling with applications to speculative decoding and lossy compression. *Advances in Neural Information Processing Systems*, 38:5576–5619, 2026.

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your llm knows the future: Uncovering its multi-token prediction potential. *arXiv preprint arXiv:2507.11851*, 2025.

Yuhao Shen, Junyi Shen, Quan Kong, Tianyu Liu, Yao Lu, and Cong Wang. Speculative decoding via hybrid drafting and rollback-aware branch parallelism. *arXiv preprint arXiv:2506.01979*, 2025.

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

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*, 2024a.

Ryan Sun, Tianyi Zhou, Xun Chen, and Lichao Sun. Spechub: Provable acceleration to multi-draft speculative decoding. *arXiv preprint arXiv:2411.05289*, 2024b.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. Spectr: Fast speculative decoding via optimal transport. *Advances in Neural Information Processing Systems*, 36:30222–30242, 2023.

Ziteng Sun, Uri Mendlovic, Yaniv Leviathan, Asaf Aharoni, Jae Hun Ro, Ahmad Beirami, and Ananda Theertha Suresh. Block verification accelerates speculative decoding. *arXiv preprint arXiv:2403.10444*, 2024c.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025a.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025b. URL https://arxiv.org/abs/2503.19786.

Rahul Krishna Thomas and Arka Pal. Global resolution: Optimal multi-draft speculative sampling via convex minimization. *arXiv preprint arXiv:2511.15898*, 2025.

Jörg Tiedemann. Parallel data, tools and interfaces in OPUS. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2214–2218, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. Opt-tree: Speculative decoding with adaptive draft tree structure. *Transactions of the Association for Computational Linguistics*, 13:188–199, 2025.

Yepeng Weng, Qiao Hu, Xujie Chen, Li Liu, Dianwen Mei, Huishi Qiu, Jiang Tian, and Zhongchao Shi. Traversal verification for speculative tree decoding. *arXiv preprint arXiv:2505.12398*, 2025.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *arXiv preprint arXiv:2401.07851*, 2024.

Yunfan Xiong, Ruoyu Zhang, Yanzeng Li, and Lei Zou. Dyspec: Faster speculative decoding with dynamic token tree structure. *World Wide Web*, 28(3):36, 2025.

Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. Decoding speculative decoding. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6460–6473, 2025.

Xiao Yang, Kai Sun, Hao Xin, Yushi Sun, Nikita Bhalla, Xiangsen Chen, Sajal Choudhary, Rongze D Gui, Ziran W Jiang, Ziyu Jiang, et al. Crag-comprehensive rag benchmark. *Advances in Neural Information Processing Systems*, 37:10470–10490, 2024.

Jinwei Yao, Kexun Zhang, Kaiqi Chen, Jiaxuan You, Zeke Wang, Binhang Yuan, and Tao Lin. Deft: Flash tree-attention with io-awareness for efficient tree-search-based llm inference. In *ICLR 2024 Workshop: How Far Are We From AGI*, 2024.

Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. Improving massively multilingual neural machine translation and zero-shot translation. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1628–1639, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.148. URL https://aclanthology.org/2020.acl-main.148.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *arXiv preprint arXiv:2309.08168*, 2023.

Ziyin Zhang, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Rui Wang, and Zhaopeng Tu. Draft model knows when to stop: A self-verification length policy for speculative decoding. *arXiv preprint arXiv:2411.18462*, 2024.

Weilin Zhao, Yuxiang Huang, Xu Han, Wang Xu, Chaojun Xiao, Xinrong Zhang, Yewei Fang, Kaihuo Zhang, Zhiyuan Liu, and Maosong Sun. Ouroboros: Generating longer drafts phrase by phrase for faster speculative decoding. *arXiv preprint arXiv:2402.13720*, 2024.

Huanran Zheng and Xiaoling Wang. Faster speculative decoding via effective draft decoder with pruned candidate tree. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9856–9868, 2025.

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. *arXiv preprint arXiv:2310.08461*, 2023.

Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. Multilingual machine translation with large language models: Empirical results and analysis, 2024. URL https://arxiv.org/abs/2304.04675.

A Orthogonal Drafting Work

Various avenues of work improve drafting or its integration with other phases without altering the tree construction. Some even fundamentally alter the draft-verify regime. We emphasize that many of these works are orthogonal to ours, and can be combined with dynamic delayed drafting to further improve throughput. We leave exploration of these ideas to future work.

Improving drafting without tree control. EAGLE (Li et al., 2024a) trains a lightweight draft model on features to propose many high-quality draft tokens per step. EAGLE-3 (Li et al., 2025) scales EAGLE with test-time compute to improve draft quality under fixed latency. Kangaroo (Liu et al., 2024a) employs self-speculation to train an adaptive layer on a subnetwork of the target model, to use as a draft model. Cacheback (Ma et al., 2025) uses an LRU cache of n-grams to accelerate drafting. Cascade drafting (Chen et al., 2024b) uses horizontal and vertical cascading, with larger drafts for crucial earlier tokens and smaller drafts for later tokens. Ouroboros (Zhao et al., 2024) drafts and verifies phrases rather than tokens. Previous work also considers retrieval (He et al., 2023), hierarchical drafting (Sun et al., 2024a), and layer skipping (Zhang et al., 2023; Elhoushi et al., 2024). Our work is agnostic to how draft candidates are produced, so we do not compare against these methods.

Pipelining and parallelism. Some hardware-aware work focuses on reducing rollbacks and pipeline stalls. Pearl (Liu et al., 2024b) reduces mutual waiting between drafting and target passes by overlapping computation with pre/post-verification pipelining, and tunes adaptive draft lengths for end-to-end latency instead of block efficiency. SpecBranch (Shen et al., 2025) improves Pearl by introducing branch-parallel speculative branches with rollback-aware hybrid drafts, explicitly trading off between parallelism and rollback costs. These are orthogonal to and can be integrated into our work, since we optimize for drafting and these optimize for scheduling once drafting is fixed.

Alternative multi-token prediction. Speculative decoding is not the only method that decodes multiple tokens per target call. Parallel decoding (Stern et al., 2018) proposes many tokens and uses parallel scoring to reduce autoregressive dependence, but does not faithfully maintain the target distribution. Medusa (Cai et al., 2024) trains multiple independent heads over the draft model to propose draft tokens, and Hydra (Ankner et al., 2024) extends on Medusa with sequential decoding. Multi-token prediction (Gloeckle et al., 2024; Samragh et al., 2025) trains auxiliary heads without a draft model to predict multiple future tokens from a single forward pass. Lookahead decoding (Fu et al., 2024) generates many disjoint n-grams in one parallelized step, and performs verification to maintain the target distribution.

B Review of OT-Based Methods

Here, we review the five OT-based verification algorithms. Each involves traversing the draft tree top-down based on its corresponding OTLP solver, as described in Section 3.2. Below, p and q denote the target and draft distributions, and we use the shorthand notation $\boldsymbol{x}_+ = \max\{\boldsymbol{x}, 0\}$ for vectors $\boldsymbol{x}$. We also use $\propto$ to denote normalizing a distribution.

B.1 NSS

NSS[3] is the simplest OT-based algorithm, explicitly defined in (Miao et al., 2024). Ignoring the draft tokens, NSS always directly samples a token from the target distribution, as in Algorithm 1. Because NSS does not depend at all on draft model probabilities, it can be used in any drafting regime. Thus, methods like EAGLE-2, which use deterministic trees, use NSS.

Algorithm 1 NSS OTLP Solver

Require: Target dist. $p \in \Delta(\mathcal{V})$, draft dist. $q \in \Delta(\mathcal{V})$, i.i.d. draft tokens $X_1, \ldots, X_k \sim q$
1: Sample $Y \sim p$
2: **return** Y

B.2 Naive

Naive speculative sampling, as we described in Section 3.1, was originally formulated as a single-path method (Leviathan et al., 2023; Chen et al., 2023). However, one can observe that independently accepting draft tree nodes, and then selecting the maximum depth node with all accepted ancestors, is equivalent to performing a top-down OT-based draft tree traversal when the draft tree is a single path. The output of the OTLP solver here is given by either accepting the single child token, or sampling from a residual. This can be extended to a multi-path method in Algorithm 2 by following the same OTLP solver only on the first draft token X_1, but still allowing one to output and traverse to other draft tokens $X_2, \ldots, X_k$ when sampling from the residual. Note that while the output of NSS is independent of all draft tokens, the output of Naive is independent of all but the first draft token.

Algorithm 2 Naive OTLP Solver (Speculative Decoding)

Require: Target dist. $p \in \Delta(\mathcal{V})$, draft dist. $q \in \Delta(\mathcal{V})$, i.i.d. draft tokens $X_1, \ldots, X_k \sim q$
1: Sample $U \sim \mathcal{U}([0, 1])$
2: **if** $U \leq p(X_1)/q(X_1)$ **then**
3: **return** X_1
4: **end if**
5: $p_{\text{res}} \propto (p - q)_+$
6: Sample $Y \sim p_{\text{res}}$
7: **return** Y

B.3 SpecTr

SpecTr uses the OTLP solver K-SEQ, described in detail by Sun et al. (2023), so we recap the sampling procedure below and forgo proofs of losslessness. First, we compute the division factor $\rho^\star$. For each ρ, define the quantities

$$\beta(\rho) = \sum_{x \in \mathcal{V}} \min\left(\rho^{-1} p(x), q(x)\right), \tag{5}$$

$$p_{\text{acc}}(\rho) = 1 - (1 - \beta(\rho))^k. \tag{6}$$

The division factor is the root of $\rho \mapsto p_{\text{acc}}(\rho) - \rho\beta(\rho)$, which is monotone decreasing on $[1, k]$, so binary search can be used to efficiently compute $\rho^\star$. Then, a $\rho^\star$-weighted version of the Naive OTLP solver is performed for multiple rounds, as in Algorithm 3. This OTLP solver precisely reduces to

[3] Miao et al. (2024) called this "Naive Speculative Sampling", but we refer to standard speculative sampling as naive, and call this NSS.

the Naive solver when $k = 1$. Note that $\rho^\star$ up to k can be increased and this will remain an OTLP solver, but the optimal acceptance rate (see Section C) is provably achieved at $\rho^\star$.

Algorithm 3 SpecTr OTLP Solver (K-SEQ)

Require: Target dist. $p \in \Delta(\mathcal{V})$, draft dist. $q \in \Delta(\mathcal{V})$, draft tokens $X_1, \ldots, X_k$
Solve $p_{\text{acc}}(\rho) = \rho\beta(\rho)$ for $\rho^\star$ by binary search
$\beta \leftarrow \sum_{t\in\mathcal{V}} \min\left(p(t)/\rho^\star,\, q(t)\right)$
$p_{\text{acc}} \leftarrow 1 - (1-\beta)^k$
$\gamma \leftarrow p_{\text{acc}}/\beta$
for $i = 1$ to k **do**
  Sample $U_i \sim \mathcal{U}([0,1])$
  if $\rho^\star U_i \le p(X_i)/q(X_i)$ **then**
    return X_i
  end if
end for
$p_{\text{res}} \propto (p - \min(p/\rho^\star, q)\cdot\gamma)_+$
Sample $Y \sim p_{\text{res}}$
return Y

B.4 SpecInfer

SpecInfer is described alongside NSS in Miao et al. (2024). Similar to SpecTr, this algorithm reduces to Naive when $k = 1$. While both SpecTr and SpecInfer involve at most k rounds of potential acceptances before sampling from the residual, SpecTr only computes the residual at the end, whereas SpecInfer updates it at each round via uniform child selection.

Algorithm 4 SpecInfer OTLP Solver

Require: Target dist. $p \in \Delta(\mathcal{V})$, draft dist. $q \in \Delta(\mathcal{V})$, draft tokens $X_1, \ldots, X_k$
$S \leftarrow [X_1, \ldots, X_k]$
while $S \neq [\,]$ **do**
  Sample $x \sim \mathcal{U}(S)$ and $U \sim \mathcal{U}([0,1])$
  if $U \le p(x)/q(x)$ **then**
    return x
  end if
  $p \propto (p - q)_+$
  Remove one occurrence of x from S
end while
Sample $Y \sim p$
return Y

B.5 Khisti

Khisti first solves a truncated OTLP based on p, q, k to generate an importance-weighted distribution r. Then, they sample a token $x \sim r$ by using pairwise tournament selection on $X_{1:k}$, and perform naive speculative sampling, with r replacing the draft q and $[x]$ replacing the draft tokens $[X_1, \ldots, X_k]$. For further details, see Section 4 of Khisti et al. (2025).

Algorithm 5 Khisti OTLP Solver

Require: Target dist. $p \in \Delta(\mathcal{V})$, draft dist. $q \in \Delta(\mathcal{V})$, draft tokens $X_1, \ldots, X_k$
$r \leftarrow$ KHISTI_IMPORTANCE_SAMPLE(p, q, k)
$x \leftarrow$ KHISTI_TOURNAMENT_SELECT$(r, X_{1:k})$
return NAIVE_OTLP_SOLVER$(p, r, [x])$

C OTLP Acceptances

Now, for each of the five OT-based verification algorithms from Section B, we show how to compute the acceptance rate as defined in Definition 5.1. We use the exact same notation. These algorithms can all be obtained by following the logic of the OTLP solver and adding the probabilities of accepting tokens at various rounds and sampling draft tokens from the residual. We have empirically confirmed their accuracy with extensive Monte Carlo sampling tests.

There is one caveat: one cannot get an exact acceptance for Khisti efficiently, as it involves an extremely large LP. However, it is possible to obtain a lower bound by simply ignoring the residual contribution for draft tokens other than x.

Algorithm 6 NSS Acceptance

Require: Target distribution $p \in \Delta(\mathcal{V})$, draft distribution $q \in \Delta(\mathcal{V})$, $k \geq 1$

return $\sum_{t\in\mathcal{V}} p(t)\big(1-(1-q(t))^k\big)$

Algorithm 7 Naive Acceptance

Require: Target distribution $p \in \Delta(\mathcal{V})$, draft distribution $q \in \Delta(\mathcal{V})$, $k \geq 1$

return $\sum_{t\in\mathcal{V}} \min(p(t), q(t)) + \sum_{t\in\mathcal{V}} (p(t)-q(t))_+ \big(1-(1-q(t))^{k-1}\big)$

Algorithm 8 SpecTr Acceptance

Require: Target distribution $p \in \Delta(\mathcal{V})$, draft distribution $q \in \Delta(\mathcal{V})$, $k \geq 1$

Solve $p_{\text{acc}}(\rho) = \rho\beta(\rho)$ for $\rho^\star$ by binary search

$\beta \leftarrow \sum_{t\in\mathcal{V}} \min\left(p(t)/\rho^\star,\ q(t)\right)$

$p_{\text{acc}} \leftarrow 1-(1-\beta)^k$

$\gamma \leftarrow p_{\text{acc}}/\beta$

$p_{\text{res}} \propto (p - \min(p/\rho^\star, q)\cdot\gamma)_+$

$r \leftarrow (q - p/\rho^\star)_+/(1-\beta)$

return $p_{\text{acc}} + (1-p_{\text{acc}}) \sum_{t\in\mathcal{V}} p_{\text{res}}(t)\big(1-(1-r(t))^k\big)$

Algorithm 9 SpecInfer Acceptance

Require: Target distribution $p \in \Delta(\mathcal{V})$, draft distribution $q \in \Delta(\mathcal{V})$, $k \geq 1$

$p_{\text{rej}} \leftarrow 1, m \leftarrow \mathbf{1}$

for $i = 1$ to k **do**

  $r \leftarrow \sum_{t\in\mathcal{V}} \min(p(t), q(t))$

  $p_{\text{rej}} \leftarrow p_{\text{rej}} \cdot (1-r)$

  $m \leftarrow m \odot (\mathbf{1} - (q-p)_+/(1-r))$

  $p \propto (p-q)_+$

end for

$\alpha \leftarrow (1-p_{\text{rej}}) + p_{\text{rej}} \sum_{t\in\mathcal{V}} p(t)\big(1-m(t)\big)$

return α

Algorithm 10 Khisti Acceptance Lower Bound

Require: Target distribution $p \in \Delta(\mathcal{V})$, draft distribution $q \in \Delta(\mathcal{V})$, $k \geq 1$

$r \leftarrow$ KHISTI_IMPORTANCE_SAMPLE(p, q, k)

return $\sum_{t\in\mathcal{V}} \min(p(t), r(t))$

D NDE is Lossless

Here, we prove that delayed expansion preserves the target distribution p for both OT-based verifiers and Traversal Verification, and then extend this guarantee to OT-based NDE and Traversal-NDE.

Theorem D.1. *Let c be the current context. Suppose the selector chooses an action $A = (K, L_1, L_2)$ using only information available before the new draft tree is sampled, and uses this action to draft the A-delayed-tree T_A at context $\mathbf{c}$. Then all valid OT-based verifiers and Traversal Verification preserve the target distribution when applied to T_A. Thus, OT-based NDE and Traversal-NDE are lossless.*

Proof. We first condition on an arbitrary fixed action $A = a$.

For an OT-based verifier, consider any node u reached during top-down verification, and let k_u be its child multiplicity. Before the branching point, $k_u = 1$ and its child token is sampled from $q(\cdot \mid u)$. After branching, the draft paths sharing u are independent samples from q, so their next unused tokens are i.i.d. $X_1, \dots, X_{k_u} \mid u \sim q(\cdot \mid u)$. Also, conditioning on verification reaching u does not change this distribution: reaching u is determined by ancestor proposals and verifier randomness, but the new proposals at u have not yet participated in verification. Now, by Definition 3.2, a valid OTLP solver satisfies $f_{p(\cdot|u),q(\cdot|u),k_u}(X_1, \dots, X_{k_u}) \sim p(\cdot \mid u)$. Thus, conditioned on every accepted prefix u, the next token emitted by verification follows the target distribution $p(\cdot \mid u)$. If that token is represented by a draft child, verification continues to that child; otherwise verification terminates with a correction token. Inducting over emitted positions shows for every r and emitted prefix $y_{1:r}$ that

$$\mathbb{P}(Y_{1:r} = y_{1:r} \mid c, A = a) = \prod_{j=1}^{r} p(y_j \mid c, y_{1:j-1}). \tag{7}$$

Thus OT-based verification on every fixed delayed-tree action is lossless.

For Traversal, we reduce directly to Weng et al. (2025). Their Definition 3.1 defines a valid stochastic sampling tree by requiring children to be generated from the draft conditional, with or without replacement, and their Theorem 3.3 proves Traversal lossless on any such tree. Thus, it suffices to show that T_a is valid. Consider any indexed root-to-node path $v = (X_0, \dots, X_d)$ in a delayed tree. Along the trunk, each token is sampled autoregressively from q; after the branching point, each continuation is independently sampled autoregressively from the same conditional draft model. Hence, we exactly satisfy the condition in Definition 3.1 of Weng et al. (2025):

$$\mathbb{P}(v \mid A = a) = \prod_{j=1}^{d} q(X_j \mid X_0, \dots, X_{j-1}). \tag{8}$$

Therefore T_a is a valid sampling tree, as desired.

Finally, we remove the requirement that $A = a$ is fixed. The selector chooses A before the new tree is sampled. Since both verifiers are lossless for every fixed action a, for any output event E, $\mathbb{P}(Y \in E \mid c, A = a) = \mathbb{P}_p(E \mid c)$. Marginalizing over the selector, $\mathbb{P}(Y \in E \mid c) = \sum_a \mathbb{P}(A = a \mid c)\mathbb{P}(Y \in E \mid c, A = a) = \mathbb{P}_p(E \mid c)$. Thus context-dependent action selection does not change the output distribution, and so OT-based NDE and Traversal-NDE are lossless. □

E OTLP Branching

Next, for the five OT-based verification algorithms from Section B, we compute branching probabilities from Definition 5.3. We again use the notation from Section B. These algorithms can similarly be obtained by following the logic of the OTLP solver and mixing the branching probabilities from round-wise token acceptances with the final residual. We have confirmed these algorithms are accurate with Monte Carlo sampling. Unlike acceptance computation, Khisti does have an exact algorithm, by extending tournament selection to a distribution on the output of tournament selection conditioned on the draft tokens.

Algorithm 11 NSS Branching

Require: Target distribution $p \in \Delta(\mathcal{V})$, draft distribution $q \in \Delta(\mathcal{V})$, draft tokens $X_1, \ldots, X_k$

return $\{X_i \mapsto p(X_i)\}$

Algorithm 12 Naive Branching

Require: Target distribution $p \in \Delta(\mathcal{V})$, draft distribution $q \in \Delta(\mathcal{V})$, draft tokens $X_1, \ldots, X_k$

$a \leftarrow \min(1, p(X_1)/q(X_1))$

$p_{\text{res}} \propto (p - q)_+$

return $\{X_i \mapsto (1-a)p_{\text{res}}(X_i) + a \cdot \mathbf{1}\{X_i = X_1\}\}$

Algorithm 13 SpecTr Branching

Require: Target distribution $p \in \Delta(\mathcal{V})$, draft distribution $q \in \Delta(\mathcal{V})$, draft tokens $X_1, \ldots, X_k$

Solve $p_{\text{acc}}(\rho) = \rho\beta(\rho)$ for $\rho^\star$ by binary search

$\beta \leftarrow \sum_{t \in \mathcal{V}} \min\left(p(t)/\rho^\star,\ q(t)\right)$

$p_{\text{acc}} \leftarrow 1 - (1-\beta)^k$

$\gamma \leftarrow p_{\text{acc}}/\beta$

$p_{\text{res}} \propto (p - \min(p/\rho^\star, q) \cdot \gamma)_+$

for $i = 1$ to k **do**

  $a_i \leftarrow \min(1, p(X_i)/(\rho^\star q(X_i)))$

end for

return $\{X_i \mapsto \sum_{j=1}^k a_j \mathbf{1}\{X_j = X_i\} \prod_{l<j}(1-a_l) + p_{\text{res}}(X_i) \prod_{l=1}^k (1-a_l)\}$

Algorithm 14 SpecInfer Branching

Require: Target distribution $p \in \Delta(\mathcal{V})$, draft distribution $q \in \Delta(\mathcal{V})$, draft tokens $X_1, \ldots, X_k$

$p_0 \leftarrow p$

for $i = 1$ to k **do**

  $p_i \propto (p_{i-1} - q)_+$

  $a_i \leftarrow \min(\mathbf{1}, p_{i-1}/q)$

end for

for $i = k$ to 0 **do**

  for multisets $S \subseteq [X_1, \ldots, X_k]$ s.t. $|S| = k - i$ **do**

    for $x \in \{X_1, \ldots, X_k\}$ **do**

      if $i = k$ **then**

        $\mathcal{B}_i(S; x) \leftarrow p_k(x)$

      else

        $\mathcal{B}_i(S; x) \leftarrow |S|^{-1} \sum_{t \in S} (a_i(t)\mathbf{1}\{t = x\} + (1 - a_i(t))\mathcal{B}_{i+1}(S \setminus \{t\}; x))$

      end if

    end for

  end for

end for

return $\{X_i \mapsto \mathcal{B}_0([X_1, \ldots, X_k]; X_i)\}$

Algorithm 15 Khisti Branching

Require: Target distribution $p \in \Delta(\mathcal{V})$, draft distribution $q \in \Delta(\mathcal{V})$, draft tokens $X_1, \ldots, X_k$

$r \leftarrow$ KHISTI_IMPORTANCE_SAMPLE(p, q, k)
for $x \in \{X_1, \ldots, X_k\}$ **do**
    $\pi_x \leftarrow \mathbb{P}(\text{KHISTI_TOURNAMENT_SELECT}(r, X_{1:k}) = x)$
end for
return $\sum_{x \in \{X_1, \ldots, X_k\}} \pi_x \cdot \text{NAIVE_BRANCHING}(p, r, [x])$

F Block Efficiency in Traversal Verification

Traversal Verification initializes node weights by following the block verification recursive update. Then, it repeatedly checks the first leaf in DFS order. If the leaf is accepted, the verifier returns it. Otherwise, the leaf is deleted, the parent distribution is replaced by a residual distribution, and the process continues until a node is accepted. For full details on the algorithm, see Weng et al. (2025).

Now, we show how to compute the expected block efficiency in Traversal Verification. Unlike OT-based verifiers, Traversal Verification does not induce a top-down traversal of the tree. Instead, for a fixed draft tree $\mathcal{T}$, the verifier checks leaves in a deterministic leaf-removal order. Conditioned on visiting a particular leaf, all prior leaves must have been rejected, so the verifier is deterministic. Thus, if $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m$ is the leaf-removal order and a_i is the conditional acceptance at $\boldsymbol{c}_i$, then

$$\mathbb{E}[\tau|\mathcal{T}] = \sum_{i=1}^{m} \text{depth}(\boldsymbol{c}_i)\, a_i \prod_{j=1}^{i-1}(1 - a_j). \tag{9}$$

The full algorithm is shown in Algorithm 16. Note that the root $\boldsymbol{c}_{\text{root}}$ in the algorithm is exactly $\boldsymbol{c}_m$, the last in the leaf-removal order. After this, Equation (1) applies in the same way as for OT-based methods to estimate the expected block efficiency $\mathbb{E}[\tau + 1] = \mathbb{E}_{\mathcal{T}}\left[\mathbb{E}[\tau + 1|\mathcal{T}]\right]$.

Algorithm 16 Expected Traversal Block Efficiency

Require: Draft tree $\mathcal{T}$, target dist. $p_{\boldsymbol{c}}(\cdot) = p(\cdot|\boldsymbol{c})$ for $\boldsymbol{c} \in \mathcal{T}$, draft dist. $q_{\boldsymbol{c}}(\cdot) = q(\cdot|\boldsymbol{c})$ for $\boldsymbol{c} \in \mathcal{T}$

Iteratively remove first leaf to get leaf-removal order $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m$
Set root node weight $w(\boldsymbol{c}_m) \leftarrow 1$
for $\boldsymbol{c} = \boldsymbol{c}_m, \ldots, \boldsymbol{c}_1$ **do**
  $p_{\text{cur}} \leftarrow p_{\boldsymbol{c}}$
  $q_{\text{cur}} \leftarrow q_{\boldsymbol{c}}$
  for each child $(\boldsymbol{c}, x)$ of $\boldsymbol{c}$ in leaf-removal order **do**
    $w((\boldsymbol{c}, x)) \leftarrow \min\{1, w(\boldsymbol{c})p_{\text{cur}}(x)/q_{\text{cur}}(x)\}$
    $p_{\text{res}} \leftarrow [w(\boldsymbol{c})p_{\text{cur}} - q_{\text{cur}}]_+$
    $r \leftarrow \sum_{t \in \mathcal{V}} p_{\text{res}}(t)$
    if $r > 0$ **then**
      $w(\boldsymbol{c}) \leftarrow r/(r + 1 - w(\boldsymbol{c}))$
      $p_{\text{cur}} \leftarrow p_{\text{res}}/r$
    else
      $w(\boldsymbol{c}) \leftarrow 0$
    end if
    $q_{\text{cur}}(x) \leftarrow 0$ and renormalize q_{cur}
  end for
  $a(\boldsymbol{c}) \leftarrow w(\boldsymbol{c})$
end for
Set root node acceptance $a(\boldsymbol{c}_m) \leftarrow 1$
return $\sum_{i=1}^{m} \text{depth}(\boldsymbol{c}_i)\, a(\boldsymbol{c}_i) \prod_{j=1}^{i-1}(1 - a(\boldsymbol{c}_j))$

G Neural Selector Details

Let $\mathcal{A}$ be the space of all supported actions for delayed expansion, defined as the Cartesian product $\mathcal{A} = \{1, ..., K_{\text{max}}\} \times \{0, ..., L_{1,\text{max}}\} \times \{0, ..., L_{2,\text{max}}\}$. An action $a = (K, L_1, L_2) \in \mathcal{A}$ specifies the branching factor K, the trunk draft length L_1, and the branch draft length L_2. In delayed tree expansion, we aim to maximize the following throughput objective, which is measured in tokens per second (TPS), over all policies π mapping from input features to $\mathcal{A}$:

$$\max_{\pi} \mathbb{E}_{\boldsymbol{c}}[\text{TPS}_\pi(\boldsymbol{c})] \quad \text{where} \quad \text{TPS}_\pi(\boldsymbol{c}) := \frac{\mathbb{E}_{a\sim\pi(\cdot|\boldsymbol{c})}[\tau + 1|\boldsymbol{c}, a]}{\mathbb{E}_{a\sim\pi(\cdot|\boldsymbol{c})}[\mathcal{W}(\boldsymbol{c}, a)]}. \tag{10}$$

Here, $\mathbb{E}[\tau + 1|\boldsymbol{c}, a]$ is the block efficiency at context $\boldsymbol{c}$ after choosing drafting action $a \in \mathcal{A}$, and $\mathbb{E}[\mathcal{W}(\boldsymbol{c}, a)]$ is the expected wall-time (in seconds) of the necessary draft and target passes performed at context $\boldsymbol{c}$ under the drafting action a. In general, for a fixed hardware and model setup, wall-time $\mathcal{W}(\boldsymbol{c}, a)$ is largely a function of $|\boldsymbol{c}|$ rather than $\boldsymbol{c}$, i.e. it depends more on context length than content. The TPS expectation is taken over the distributions of contexts $\boldsymbol{c}$ we would observe across all inputs. To solve this optimization problem, we define a selector policy with the following input features:

- **Target and draft hidden state features.** These are h^p_{prev}, h^q_{prev} for p, q at the previous token, and only h^q_{cur} at the current token[4]. We perform an extra draft forward pass to get h^q_{cur} pre-drafting, which adds some latency. All these are projected to a shared dimension d.
- **Scalar uncertainty features.** These include the entropies $H(p_{\text{prev}}), H(q_{\text{prev}}), H(q_{root})$, local divergences $D_{\text{KL}}(p_{\text{prev}} \| q_{\text{prev}}), D_{\text{KL}}(q_{\text{prev}} \| p_{\text{prev}})$, and a local L_1 distance $\|p_{\text{prev}} - q_{\text{prev}}\|_1$.
- **Local parameter features.** These include the context length $|\boldsymbol{c}|$ and the sampling hyperparameters, which are target temperature and target nucleus sampling threshold p_{top}.
- **Latency features.** These include runtimes for a draft and target forward pass at the current context length, estimated from warmup runs, to capture hardware-dependent effects.

We design this selector as an MLP policy with separate projections for each hidden-state feature block. Let ϕ_1, ϕ_2, ϕ_3 be the linear projections from each representation vector to a shared dimension $d = 128$, each followed by layer normalization (LN). Also, let $s(x)$ be the standardized scalar feature vector, concatenating uncertainty, parameter, and latency features. The policy then computes logits

$$z_\theta(x) = \text{MLP}\left(\left[\text{LN}(\phi_1(h^p_{\text{prev}})), \text{LN}(\phi_2(h^q_{\text{prev}})), \text{LN}(\phi_3(h^q_{\text{cur}})), s(x)\right]\right). \tag{11}$$

Our MLP is lightweight, with two hidden layers of sizes 512 and 32, GELU activations (Hendrycks and Gimpel, 2023), and dropout. The output layer of this model has dimension $|\mathcal{A}|$, as the logits represent the probability that each action in $\mathcal{A}$ is optimal for throughput. Using this structure, the overhead for the selector forward pass is negligible compared drafting or the target model pass. In fact, in our experiments, we find the overhead from the neural selector is under 0.5% of end-to-end wall-time. The MLP forward pass latency is actually constant with respect to context length, since our feature sizes are not context-dependent, which makes it promising for long-context decoding.

Let $t_p(l)$ be the measured target model time in seconds for a forward pass at context length l, and let $t_q(l)$ be the draft model time. Then, for an action $a = (K, L_1, L_2)$ at context length $|\boldsymbol{c}|$ we approximate total wall-clock time by adding trunk drafting, branched drafting, and target pass times:

$$\mathcal{W}(\boldsymbol{c}, a) \approx \underbrace{\sum_{j=0}^{L_1-1} t_q(|\boldsymbol{c}| + j)}_{\text{trunk drafting}} + \underbrace{\sum_{j=0}^{L_2-1} t_q(|\boldsymbol{c}| + L_1 + jK)}_{\text{branch drafting}} + \underbrace{t_p(|\boldsymbol{c}| + L_1 + KL_2)}_{\text{target forward pass}}. \tag{12}$$

We now plug this into Equation (10) to estimate $\text{TPS}(\pi)$. We define our minimization objective as below in Equation (13), where B is the minibatch size, λ is the penalty weight for regressing below the baseline, and $\mathcal{I}_\alpha$ are the top-α-fraction indices of penalty terms $\max\{1 - \text{TPS}_\pi(\boldsymbol{c}_i)/\text{TPS}_{\text{base}}(\boldsymbol{c}_i), 0\}$

[4]We cannot use target features here as the root token is missing a KV cache row from the last iteration, which requires an extra target forward pass. This is also the case for drafting, but an extra draft forward pass is relatively cheap, so we perform it.

in the minibatch $\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_B\}$, and TPS_{base} is throughput for the static baseline we want to improve:

$$\mathcal{L} = \frac{1}{B}\sum_{i=1}^{B}\left(-\log\frac{\text{TPS}_\pi(\boldsymbol{c}_i)}{\text{TPS}_{\text{base}}(\boldsymbol{c}_i)}\right) + \frac{\lambda}{|\mathcal{I}_\alpha|}\sum_{i\in\mathcal{I}_\alpha}\max\left(1-\frac{\text{TPS}_\pi(\boldsymbol{c}_i)}{\text{TPS}_{\text{base}}(\boldsymbol{c}_i)}, 0\right)^2. \tag{13}$$

This additional term penalizes worst-case throughput regressions relative to the baseline, to prevent the policy from trading occasional but large regressions for marginal average improvements.

To train the selector as described above, we construct a target signal for each action from an offline dataset of speculative decoding traces, taking a root every $N = 16$ tokens. This follows our approach in Section 4.1. For each root $\boldsymbol{c}$ and action a, we store an empirical estimate of the block efficiency $\mathbb{E}[\tau + 1]$ of an a-delayed tree at $\boldsymbol{c}$, by using either Equation (2) for OT-based methods or the methodology in Section F for Traversal Verification. We also estimate the wall-times $\mathcal{W}(\boldsymbol{c}, a)$ by using latency measurements derived from a microbenchmark warm-up run, then approximate the total wall-clock time for all necessary forward passes t_q, t_l as shown in Equation (12). With these estimates, we can compute an estimate of $\text{TPS}_\pi(\boldsymbol{c})$. For each training example, we also have a static baseline configuration a_i^{base} associated with its sampling parameters, so we can use these to compute the context-dependent baseline $\text{TPS}_{\text{base}}(\boldsymbol{c})$. Then, Equation (13) gives the training loss.

In practice, after hyperparameter tuning, we use learning rate 4×10^{-5}, weight decay 0.05, and we perform full SFT for 10 epochs with AdamW. We use a batch size of 512, dropout of 0.15, penalty weight of $\lambda = 2$, penalty fraction of $\alpha = 0.1$, hidden feature dropout of 0.5, and hidden feature noise standard deviation of 0.02.

In Algorithm 17, we explicitly describe how our neural delay-and-expand (NDE) selector, once trained, can be integrated into the three-step speculative decoding process. We note that the verification algorithm $\mathcal{A}$ does *not* have to match the verification algorithm used to train NDE, to maintain a valid speculative decoding algorithm. However, with this verification mismatch, throughput may degrade.

Algorithm 17 Full NDE Algorithm

Require: target model M_p, draft model M_q, context $\boldsymbol{c}$, neural selector π_θ, verification algorithm $\mathcal{A}$.

while $|\boldsymbol{c}| <$ max_tokens **do**
  Run neural selector to obtain categorical policy $\pi_\theta(\cdot|\boldsymbol{c})$ as in Section G
  Select expansion parameters $(K, L_1, L_2) \leftarrow \arg\max_{a\in\mathcal{A}} \pi_\theta(a \mid c)$ as in Section 6
  Sample a (K, L_1, L_2)-delayed tree $\mathcal{T} \sim M_q(\cdot \mid c)$ in Definition 5.2, save $q(\cdot \mid u)$ for $u \in \mathcal{T}$
  Run one batched target pass with M_p on $\mathcal{T}$ and $\mathcal{T}$-tree mask, save $p(\cdot \mid u)$ for $u \in \mathcal{T}$
  Verify and update $\boldsymbol{c} \leftarrow \mathcal{A}\left(\mathcal{T}, \{p(\cdot \mid u), q(\cdot \mid u)\}_{u\in\mathcal{T}}\right)$ as in Section 3.2
end while

H Extended Online Experimental Results

H.1 Qwen Datasets

Table 5: All algorithm throughputs (tokens per second) for each dataset in Qwen.

	Writing	Coding	Translation	Math (E)	Math (H)
Khisti, delayed expansion	14.21	17.81	14.16	23.05	22.80
Khisti	11.22	19.87	13.88	24.29	24.07
NaiveTree, delayed expansion	14.57	21.91	16.29	28.89	26.69
NaiveTree	10.25	19.08	13.28	27.52	25.67
Naive	7.90	17.47	9.76	24.84	22.77
NSS, delayed expansion	11.85	20.61	16.93	26.96	25.10
NSS	8.00	16.28	12.67	24.68	22.76
SpecInfer, delayed expansion	15.52	23.16	17.39	29.31	27.33
SpecInfer	10.77	19.76	13.43	27.81	26.14
SpecTr, delayed expansion	15.58	22.03	17.26	27.33	26.50
SpecTr	10.94	19.31	13.29	26.56	24.96
BV	9.53	18.24	11.22	24.69	22.81
Traversal, $K = 2$	11.11	20.47	12.94	27.44	25.56
Traversal, $K = 3$	11.47	21.34	14.23	29.05	26.63
Traversal, $K = 4$	11.95	22.90	14.97	29.41	27.49

Table 6: All algorithm block efficiencies (tokens per call) for each dataset in Qwen.

	Writing	Coding	Translation	Math (E)	Math (H)
Khisti, delayed expansion	2.56	5.21	3.22	7.04	6.81
Khisti	2.78	5.16	3.50	6.59	6.50
NaiveTree, delayed expansion	2.41	5.31	3.42	7.06	6.49
NaiveTree	2.62	5.14	3.63	7.00	6.60
Naive	2.18	4.91	2.73	6.91	6.30
NSS, delayed expansion	2.00	4.48	3.12	6.24	5.74
NSS	2.06	4.40	3.46	6.34	5.93
SpecInfer, delayed expansion	2.64	5.39	3.29	7.20	6.61
SpecInfer	2.79	5.33	3.67	7.12	6.76
SpecTr, delayed expansion	2.69	5.32	3.37	6.87	6.59
SpecTr	2.85	5.33	3.69	7.04	6.65
BV	2.31	4.48	2.78	6.02	5.59
Traversal, $K = 2$	2.70	5.09	3.22	6.63	6.20
Traversal, $K = 3$	2.83	5.38	3.60	7.10	6.51
Traversal, $K = 4$	2.99	5.80	3.83	7.24	6.78

H.2 Qwen Sampling Configurations

Table 7: All algorithm throughputs (tokens per second) for each sampling configuration in Qwen.

	Temperature (top-p=1)						Top-p	
	0.2	0.4	0.6	0.8	1.0	1.2	0.90	0.99
Khisti, delayed expansion	19.59	19.94	20.06	19.42	17.36	13.58	19.50	17.80
Khisti	18.23	18.66	18.90	19.07	19.37	18.53	18.11	18.47
NaiveTree, delayed expansion	22.17	22.41	22.52	22.01	21.47	19.28	21.88	21.65
NaiveTree	19.75	20.20	19.82	19.52	18.98	16.90	19.03	19.06
Naive	15.40	15.86	16.58	16.85	16.74	16.05	17.51	17.39
NSS, delayed expansion	22.56	22.60	21.88	21.07	19.57	15.93	19.21	19.49
NSS	19.31	18.97	18.55	17.76	15.93	12.72	15.77	16.04
SpecInfer, delayed expansion	22.59	22.67	22.68	22.94	22.82	21.06	22.69	22.87
SpecInfer	19.81	19.99	20.17	20.07	19.61	17.67	19.72	19.62
SpecTr, delayed expansion	21.06	21.88	22.15	22.47	22.35	20.21	21.89	21.93
SpecTr	18.70	19.49	19.72	19.25	19.14	17.19	19.15	19.46
BV	16.07	17.24	17.64	17.81	17.90	15.91	17.96	17.84
Traversal, K=2	18.93	19.61	20.14	20.20	19.83	17.99	19.68	19.65
Traversal, K=3	20.14	20.78	20.88	21.18	20.87	18.68	20.93	20.90
Traversal, K=4	21.09	21.56	21.90	21.63	21.52	19.79	21.68	21.60

Table 8: All algorithm block efficiences (tokens per call) for each sampling configuration in Qwen.

	Temperature (top-p=1)						Top-p	
	0.2	0.4	0.6	0.8	1.0	1.2	0.90	0.99
Khisti, delayed expansion	4.85	5.02	5.21	5.42	5.28	3.91	4.88	5.18
Khisti	5.13	5.25	5.37	5.28	4.51	3.40	5.09	5.21
NaiveTree, delayed expansion	5.15	5.14	5.25	5.02	4.86	4.26	4.95	4.89
NaiveTree	5.16	5.20	5.17	5.11	4.97	4.44	4.98	4.96
Naive	4.48	4.65	4.84	4.93	4.61	4.18	4.59	4.56
NSS, delayed expansion	5.01	5.01	4.73	4.47	4.05	3.09	4.05	4.09
NSS	5.05	4.98	4.89	4.65	4.19	3.38	4.16	4.22
SpecInfer, delayed expansion	5.11	5.12	5.11	5.13	5.13	4.53	5.01	5.06
SpecInfer	5.18	5.23	5.27	5.27	5.15	4.64	5.17	5.15
SpecTr, delayed expansion	4.79	5.12	5.09	5.20	5.19	4.40	4.96	4.99
SpecTr	5.04	5.24	5.28	5.20	5.15	4.62	5.14	5.21
BV	3.96	4.20	4.30	4.36	4.37	3.93	4.39	4.38
Traversal, K=2	4.65	4.78	4.89	4.93	4.83	4.40	4.84	4.82
Traversal, K=3	4.98	5.13	5.17	5.28	5.17	4.62	5.17	5.15
Traversal, K=4	5.25	5.38	5.40	5.43	5.40	4.97	5.40	5.40

H.3 Gemma Datasets

Table 9: All algorithm throughputs (tokens per second) for each dataset in Gemma.

	Writing	Coding	Translation	Math (E)	Math (H)
Khisti, delayed expansion	8.93	9.90	10.73	17.39	17.28
Khisti	3.53	5.08	8.88	6.60	5.58
NaiveTree, delayed expansion	9.95	8.45	11.95	15.84	18.10
NaiveTree	3.48	5.26	9.74	6.48	5.04
Naive	4.72	6.79	7.30	13.15	14.07
NSS, delayed expansion	10.17	6.39	11.89	15.82	17.76
NSS	3.50	5.34	9.87	6.13	4.85
SpecInfer, delayed expansion	10.02	9.85	12.10	16.53	17.77
SpecInfer	3.59	5.34	9.92	6.55	5.67
SpecTr, delayed expansion	10.00	9.87	12.03	16.24	17.97
SpecTr	3.54	5.38	9.80	6.44	5.26
BV	5.82	7.66	8.57	15.58	15.25
Traversal, $K = 2$	6.31	8.01	9.07	14.24	16.08
Traversal, $K = 3$	6.51	8.55	9.46	15.09	17.21
Traversal, $K = 4$	6.56	8.94	9.73	15.35	17.23

Table 10: All algorithm block efficiencies (tokens per call) for each dataset in Gemma.

	Writing	Coding	Translation	Math (E)	Math (H)
Khisti, delayed expansion	2.01	3.51	2.69	5.72	5.53
Khisti	1.21	1.77	3.09	2.26	1.92
NaiveTree, delayed expansion	1.89	2.75	2.82	4.62	5.00
NaiveTree	1.17	1.78	3.25	2.19	1.68
Naive	1.74	2.60	2.80	4.91	5.17
NSS, delayed expansion	1.79	2.71	2.75	4.69	4.75
NSS	1.17	1.81	3.28	2.04	1.63
SpecInfer, delayed expansion	2.00	2.84	2.74	4.66	4.87
SpecInfer	1.20	1.80	3.27	2.17	1.90
SpecTr, delayed expansion	2.02	2.73	2.77	4.64	5.02
SpecTr	1.20	1.84	3.30	2.16	1.77
BV	1.79	2.45	2.76	4.76	4.73
Traversal, $K = 2$	2.00	2.67	3.02	4.64	5.17
Traversal, $K = 3$	2.08	2.84	3.20	4.96	5.52
Traversal, $K = 4$	2.12	3.02	3.29	5.05	5.56

H.4 Gemma Sampling Configurations

Table 11: All algorithm throughputs (tokens per second) for each sampling configuration in Gemma.

	Temperature (top-p=1)						Top-p	
	0.2	0.4	0.6	0.8	1.0	1.2	0.90	0.99
Khisti, delayed expansion	12.73	12.52	13.36	12.76	12.96	12.76	12.79	12.87
Khisti	5.95	5.87	5.86	5.82	6.01	6.00	5.98	5.95
NaiveTree, delayed expansion	13.36	12.97	12.55	12.53	12.75	12.93	12.94	12.82
NaiveTree	6.03	5.96	5.86	6.01	6.10	5.94	6.01	6.11
Naive	8.99	9.19	9.37	9.45	9.06	9.04	9.25	9.29
NSS, delayed expansion	12.72	12.38	12.54	12.47	12.04	12.40	12.05	12.64
NSS	5.95	5.96	6.31	5.84	5.83	5.88	5.88	5.86
SpecInfer, delayed expansion	13.58	13.47	13.25	13.06	13.04	13.25	13.29	13.11
SpecInfer	6.13	6.20	6.21	6.09	6.35	6.24	6.27	6.21
SpecTr, delayed expansion	13.74	12.99	13.36	13.16	12.80	13.05	13.55	13.13
SpecTr	5.72	5.91	5.77	6.44	6.25	6.05	6.21	6.34
BV	10.44	10.34	10.80	10.84	10.32	10.92	10.43	10.49
Traversal, K=2	10.99	10.96	10.71	10.53	10.80	10.40	10.74	10.81
Traversal, K=3	11.44	11.09	12.03	11.11	11.31	11.88	11.27	11.32
Traversal, K=4	12.08	12.01	11.75	11.49	11.16	11.64	11.16	11.22

Table 12: All algorithm block efficiences (tokens per call) for each sampling configuration in Gemma.

	Temperature (top-p=1)						Top-p	
	0.2	0.4	0.6	0.8	1.0	1.2	0.90	0.99
Khisti, delayed expansion	3.82	3.78	4.06	3.90	3.91	3.89	3.86	3.92
Khisti	2.05	2.04	2.02	2.01	2.07	2.06	2.09	2.06
NaiveTree, delayed expansion	3.51	3.41	3.34	3.34	3.43	3.51	3.37	3.43
NaiveTree	2.02	2.00	1.97	2.02	2.04	1.98	2.04	2.04
Naive	3.47	3.34	3.40	3.53	3.50	3.57	3.37	3.37
NSS, delayed expansion	3.50	3.40	3.38	3.38	3.18	3.26	3.25	3.37
NSS	2.01	1.98	2.09	1.95	1.97	1.95	1.97	1.97
SpecInfer, delayed expansion	3.44	3.40	3.42	3.45	3.42	3.51	3.36	3.38
SpecInfer	2.05	2.06	2.06	2.05	2.10	2.08	2.08	2.08
SpecTr, delayed expansion	3.47	3.35	3.50	3.44	3.39	3.56	3.42	3.37
SpecTr	1.94	1.99	1.94	2.16	2.11	2.06	2.11	2.11
BV	3.27	3.27	3.37	3.37	3.24	3.40	3.24	3.24
Traversal, K=2	3.58	3.55	3.46	3.49	3.49	3.45	3.49	3.49
Traversal, K=3	3.71	3.62	3.93	3.70	3.68	3.93	3.68	3.68
Traversal, K=4	3.96	3.97	3.83	3.82	3.65	3.90	3.67	3.67

H.5 Llama Datasets

Table 13: All algorithm throughput (tokens per second) for each dataset in Llama.

	Writing	Coding	Translation	Math (E)	Math (H)
Khisti, delayed expansion	11.00	14.24	13.93	13.75	13.26
Khisti	11.71	14.25	14.23	14.18	13.92
NaiveTree, delayed expansion	11.31	15.66	15.04	14.76	13.29
NaiveTree	11.77	16.10	15.71	15.68	14.28
Naive	8.65	13.05	12.72	12.42	11.04
NSS, delayed expansion	9.77	14.75	14.39	12.25	11.10
NSS	10.08	15.36	15.03	12.86	11.23
SpecInfer, delayed expansion	11.80	15.77	15.08	14.79	13.90
SpecInfer	12.29	16.43	15.77	15.52	14.27
SpecTr, delayed expansion	11.73	15.59	14.85	15.04	13.92
SpecTr	12.21	16.23	15.63	15.70	14.55
BV	9.57	13.38	12.94	13.07	12.04
Traversal, $K = 2$	10.90	14.81	14.17	14.74	13.84
Traversal, $K = 3$	11.95	15.39	14.66	14.55	13.86
Traversal, $K = 4$	12.35	15.74	15.19	15.39	15.18

Table 14: All algorithm block efficiencies (tokens per call) for each dataset in Llama.

	Writing	Coding	Translation	Math (E)	Math (H)
Khisti, delayed expansion	5.59	7.48	7.24	7.06	6.85
Khisti	5.26	6.64	6.63	6.57	6.35
NaiveTree, delayed expansion	5.15	7.36	6.98	6.75	6.10
NaiveTree	5.18	7.19	7.01	6.88	6.24
Naive	4.08	6.33	6.12	5.85	5.22
NSS, delayed expansion	4.45	7.01	6.68	5.61	5.11
NSS	4.45	6.86	6.69	5.69	4.93
SpecInfer, delayed expansion	5.39	7.57	6.99	6.81	6.43
SpecInfer	5.39	7.31	7.00	6.84	6.21
SpecTr, delayed expansion	5.43	7.58	7.04	7.08	6.58
SpecTr	5.40	7.36	7.07	7.08	6.45
BV	4.18	5.93	5.78	5.71	5.25
Traversal, $K = 2$	4.90	6.71	6.48	6.59	6.17
Traversal, $K = 3$	5.40	7.10	6.78	6.57	6.31
Traversal, $K = 4$	5.62	7.36	7.07	6.96	6.86

H.6 Llama Sampling Configurations

Table 15: All algorithm throughputs (tokens per second) for each sampling configuration in Llama.

	Temperature (top-p=1)						Top-p	
	0.2	0.4	0.6	0.8	1.0	1.2	0.90	0.99
Khisti, delayed expansion	13.20	13.14	13.56	13.44	13.52	11.44	13.91	13.67
Khisti	13.52	13.46	13.82	13.89	13.79	12.75	14.06	13.99
NaiveTree, delayed expansion	13.79	14.11	13.99	14.13	14.24	13.56	14.05	14.22
NaiveTree	14.22	14.39	14.41	15.32	14.50	14.34	15.35	15.12
Naive	10.61	11.09	11.57	11.96	12.08	11.38	12.06	11.87
NSS, delayed expansion	13.44	13.43	13.31	12.77	11.90	11.01	11.69	12.05
NSS	13.77	13.36	13.36	13.57	11.93	12.01	12.69	12.59
SpecInfer, delayed expansion	13.88	14.10	14.50	14.48	14.51	14.05	14.24	14.38
SpecInfer	13.97	14.30	14.59	15.65	14.74	15.05	15.12	15.42
SpecTr, delayed expansion	13.92	13.94	14.57	14.60	14.59	13.47	14.18	14.55
SpecTr	14.11	14.27	14.30	15.33	14.68	15.22	15.45	15.54
BV	10.97	11.63	11.98	12.55	12.79	12.24	12.74	12.70
Traversal, K=2	12.91	12.83	13.87	13.69	14.01	14.03	14.19	14.01
Traversal, K=3	13.62	14.02	13.83	14.17	14.18	14.31	14.25	14.27
Traversal, K=4	13.97	14.52	15.04	15.01	14.77	15.07	14.99	14.79

Table 16: All algorithm block efficiences (tokens per call) for each sampling configuration in Llama.

	Temperature (top-p=1)						Top-p	
	0.2	0.4	0.6	0.8	1.0	1.2	0.90	0.99
Khisti, delayed expansion	6.49	6.52	6.77	6.97	7.32	6.59	6.88	7.19
Khisti	6.58	6.58	6.75	6.85	5.72	4.25	6.78	6.81
NaiveTree, delayed expansion	6.34	6.54	6.46	6.50	6.60	6.27	6.50	6.55
NaiveTree	6.42	6.51	6.51	6.61	6.56	6.20	6.65	6.52
Naive	5.06	5.30	5.54	5.70	5.73	5.41	5.76	5.68
NSS, delayed expansion	6.24	6.24	6.21	5.94	5.49	5.06	5.46	5.56
NSS	6.26	6.05	6.06	5.82	5.41	5.20	5.49	5.48
SpecInfer, delayed expansion	6.42	6.60	6.71	6.72	6.76	6.53	6.68	6.69
SpecInfer	6.35	6.46	6.60	6.71	6.63	6.45	6.53	6.66
SpecTr, delayed expansion	6.61	6.58	7.07	6.91	6.88	6.35	6.69	6.87
SpecTr	6.50	6.56	6.60	6.69	6.76	6.66	6.78	6.82
BV	4.81	5.11	5.29	5.54	5.60	5.39	5.64	5.59
Traversal, K=2	5.83	5.79	6.23	6.19	6.29	6.30	6.41	6.31
Traversal, K=3	6.21	6.37	6.39	6.47	6.50	6.51	6.49	6.50
Traversal, K=4	6.40	6.70	6.86	6.84	6.80	6.90	6.88	6.83

I Improving Llama

We show how to improve our Llama numbers by making simple modifications to our NDE architecture. We apply dropout regularization and Gaussian noise to hidden state features to disincentivize overfitting on Llama's larger hidden state. Also, we restrict hidden state features to the target and draft model states at the token preceding the root token, excluding draft states at the root token. This last choice is further corroborated by feature subset ablations in Section J. With these changes, our model achieves comparable performance on Gemma and Qwen, while also improving Llama performance.

In Table 17, we show throughput ratio improvements of NDE over baseline algorithms, across various datasets. We also report prompt-level standard errors for the original and improved Llama throughput ratios, alongside Qwen and Gemma, in Table 18. Now, Llama is able to improve the baseline on average for all methods. In particular, we see strongest improvements with NSS for all datasets. The most consistent improvements across all methods occur in writing: for other datasets, only NSS shows meaningful improvements. We hypothesize this is because NSS baselines are particularly volatile for Llama, so NDE has more room to improve. We would not expect other OT-based algorithms to suffer from this pitfall, because the Llama draft is only around $10\times$ smaller than the target, so p and q are likely well-aligned and acceptances can remain stable across contexts; this argument does not apply to NSS as it does not use q probabilities.

Table 17: Throughput ratio improvement of our NDE approach over the baseline algorithms for Llama, with splits across writing, coding, translation, and easy and hard math shown.

Method	Writing	Coding	Translation	Math (E)	Math (H)	Average
Khisti	1.07	1.04	1.02	0.99	0.98	1.02
NaiveTree	1.15	1.04	1.02	0.99	1.04	1.05
NSS	1.21	1.04	1.05	1.12	1.20	1.12
SpecInfer	1.10	1.02	1.02	1.02	1.06	1.04
SpecTr	1.10	1.03	1.02	1.00	1.03	1.03

Table 18: Throughput ratio improvements of NDE over the corresponding tuned static baselines (mean $\pm$ prompt-level standard error), averaged across datasets and sampling configurations. Original Llama corresponds to Table 3, while improved Llama uses the modifications from this section.

Method	Qwen	Gemma	Original Llama	Improved Llama
Khisti NDE	0.99 ± 0.016	2.17 ± 0.084	0.97 ± 0.013	1.02 ± 0.011
NaiveTree NDE	1.31 ± 0.015	1.40 ± 0.102	1.21 ± 0.013	1.05 ± 0.013
NSS NDE	1.20 ± 0.019	2.09 ± 0.102	0.96 ± 0.021	1.12 ± 0.022
SpecInfer NDE	1.15 ± 0.014	2.14 ± 0.100	0.96 ± 0.013	1.04 ± 0.013
SpecTr NDE	1.14 ± 0.013	2.17 ± 0.093	0.96 ± 0.013	1.03 ± 0.013

J NDE Ablations

We now perform ablations on key NDE features and design chocies. We ablate one component at a time starting from the default configuration, which includes all hidden state features (the target and draft model states at the token preceding the root, as well as the draft states at the root token), feature dropout of 0.50, Gaussian noise with a standard deviation of 0.02, and a penalty term weight of 2.00. For hidden state features, we ablate all 8 subsets of our three feature vectors.

We report the relative throughput changes, i.e. the ratio of tokens per second after ablation relative to the default configuration, in Table 19. Surprisingly, we find that NDE performs well even without any hidden state features. We hypothesize this is because our scalar features contain sufficient information to guide dynamic parameter selection. Similarly, changes in dropout, noise, and penalty weight do not significantly impact performance. Overall, NDE performance is quite robust to these modifications.

Table 19: For all models, we ablate over subsets of hidden features, dropout, noise, and penalty term weight. Here, we show relative changes in throughput relative to the default configuration.

Ablation	Llama	Gemma	Qwen
No hidden state features	-0.1%	-2.3%	0.1%
Only {`p_feat_prev`}	0.1%	-1.0%	-1.1%
Only {`q_feat_prev`}	0.2%	-1.1%	0.9%
Only {`q_feat_cur`}	0.8%	-1.6%	0.2%
Only {`p_feat_prev`, `q_feat_prev`}	-0.5%	-0.2%	-3.0%
Only {`p_feat_prev`, `q_feat_cur`}	1.2%	2.6%	-1.9%
Only {`q_feat_prev`, `q_feat_cur`}	-0.1%	-3.5%	-3.3%
Hidden states dropout $= 0$	-0.8%	0.7%	-0.6%
Hidden states noise $= 0$	-0.1%	-1.7%	-0.6%
Penalty term weight $= 0$	-1.2%	2.4%	-0.3%

K Dynamic Drafting Baselines

AdaEDL (Agrawal et al., 2024b) performs early stopping in single-path speculative decoding. It stops drafting tokens when a score $1 - (\lambda H(q))^{1/2}$ based on the draft entropy $H(q)$ drops below a dynamic threshold, where λ is a fixed hyperparameter. This means when naively applied to our setting, AdaEDL only chooses the parameter L_1 and drafts a single path instead of a delayed tree. To adapt it to delayed drafting, we choose K, L_2 to be static based on the optimal SpecInfer configurations for $L_1 = 0, 1, 2$, which we observe AdaEDL selects over 95% of the time. Then we use SpecInfer as our verification algorithm on the delayed tree. Our results are shown in Table 20. While AdaEDL and the SpecInfer baseline each perform better in some settings, e.g. AdaEDL wins in translation and writing and SpecInfer wins in coding and math, SpecInfer NDE consistently beats both in throughput.

Table 20: Block efficiency and throughput improvements over Naive baseline for NDE and AdaEDL.

Dataset	**Block Efficiency** (tokens/call)		**Throughput** (tokens/sec)	
	Naive NDE	**AdaEDL**	**Naive NDE**	**AdaEDL**
Translation	0.90	1.09	1.29	1.15
Math (H)	0.98	0.74	1.05	0.81
Coding	1.01	0.88	1.17	0.94
Writing	0.95	1.32	1.44	1.39
Math (E)	1.01	0.72	1.05	0.77

Sequoia (Chen et al., 2024a) has an offline stage, where a static global acceptance vector is formed from calibration data and we compute target and draft runtime estimates, and an online stage, where this data is used to choose a static draft tree shape and run a sampling-without-replacement variant of SpecInfer. The shape is chosen based on a block efficiency estimator, as in our work. We follow their official codebase.[5] On the data side, because our NDE is trained across data from 8 sampling settings, we test two Sequoia variants. Sequoia-G builds a fixed shape across all sampling settings; Sequoia-S builds a fixed shape for each sampling setting, so it can better specialize. For fair comparison to data-agnostic NDE, we do not test variants with a separate Sequoia tree per dataset.

We compare Sequoia to Naive and Naive NDE, which is the closest algorithmic equivalent to the verification scheme used by Sequoia. The block efficiency and throughput results per dataset are shown in Table 21. While Sequoia-G/S and Naive each perform better in different settings (translation and writing for the former, coding and math for the latter), Naive NDE is nearly always the best. Sequoia performs significantly worse than NDE on math and coding, though it is closer on writing and translation. This highlights a potential limitation of Sequoia, since the strong speedup results in their paper only involve calibration data and evaluation on C4, OpenWebText, CNN DailyMail and MT Bench, rather than complex reasoning tasks. In additional runs, we did not observe calibrating on the same data setup as Sequoia meaningfully improved performance on these 5 datasets.

Table 21: Block efficiency and throughput comparisons between Naive, Naive NDE, and Sequoia.

Dataset	**Block Efficiency** (tokens/call)				**Throughput** (tokens/sec)			
	Seq. G	**Seq. S**	**Naive**	**Naive NDE**	**Seq. G**	**Seq. S**	**Naive**	**Naive NDE**
Translation	3.20	2.94	2.73	3.42	10.25	10.09	9.76	16.29
Coding	3.63	3.22	5.33	5.39	11.48	11.07	19.76	23.16
Math (E)	3.88	3.37	6.91	7.06	13.74	12.84	24.84	28.89
Math (H)	3.88	3.38	6.30	6.49	12.17	11.48	22.77	26.69
Writing	2.87	2.71	2.18	2.41	9.00	9.64	7.90	14.57

We note that AdaEDL and Sequoia both perform far better relatively on writing and translation than math and coding. Thus, NDE is also more robust to data distribution changes than simple entropy-based expansion rules. This matches up with our results in Section 7.2, where we find NDE has strong generalization to out-of-domain tasks.

[5]While Sequoia adds CUDA graphs, offloading and quantization in a later codebased, many of these optimizations can also be applied to NDE. Also, these optimizations do not impact our block efficiency comparisons.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: We conduct experiments in Section 4 to validate our claims about Traversal Verification and OT-based methods. We also conduct experiments in Sections 5 and 7 to motivate delayed expansion and show the efficacy of our NDE approach.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: We discuss how our NDE approach offers limited improvements for Llama, and diagnose why this happens.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [Yes]

Justification: We only provide a losslessness theorem (Section D) and provide a full proof.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: We describe all data and experiments in detail sufficient to reproduce our results.

   Guidelines:

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

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

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

Answer: [No]

Justification: We describe all data and experiments in detail sufficient to reproduce our results, and we will release our code soon.

Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: We specify all training and evaluation details in our experiments section and the appendices.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: We report prompt-level standard errors for throughput ratio improvements in Table 18.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: We provide exact details on datasets and split sizes, models, and hardware that we conduct our experiments on.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and confirmed we meet all guidelines.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: Our work involves accelerating LLM inference. This falls under foundational research, and it is not tied to any particular application, so we do not feel the need to discuss broader societal impacts.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: We do not release any new data or models, and only use public data and open-source models in our experiments.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: We cite all models, datasets, and tools we use for experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [N/A]

    Justification: We do not release any new assets.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: We do not use crowdsourcing or human subjects.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: We do not use crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: We do not use LLMs except for basic formatting and editing.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.