TSUBAME: TREE REPLAY FOR DIFFUSION-BASED SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Context-aware dynamic trees allocate the speculative decoding budget according to draft path probabilities, adapting their depth and branching to the current context. Under stochastic decoding, however, we find that this structural advantage does not always compensate for the acceptance gains of random sampling paired with advanced verification, and such dynamic trees can fall behind sampled chains in some settings. These trees grow their topology from the candidates themselves, so the tokens submitted for verification are typically the deterministic high-score tokens selected during construction. This coupling is not inherent: once the topology is fixed, its nodes can be repopulated by sampling, allowing dynamic trees to retain their structural advantage while also benefiting from random sampling and advanced verification. Diffusion-based drafters make this practical, as their parallel outputs or lightweight conditional corrections allow candidates to be regenerated cheaply after the complete topology is known. We introduce **Tsubame**, a two-pass tree speculative decoding framework for diffusion-based drafters. The first pass plans and freezes a context-aware topology using draft path scores; the second replays the fixed topology, sampling the tokens that populate its nodes to form the candidate tree for verification. We prove that Tsubame is lossless under compatible sampling and verification strategies. Experiments across three diffusion-based drafters, six datasets, and multiple candidate budgets show that Tsubame improves acceptance length and throughput over deterministic trees, including settings where it reverses their disadvantage against sampled chains.

1 INTRODUCTION

Autoregressive language models generate text one token at a time, requiring a separate forward pass for each new token. Speculative decoding reduces this sequential bottleneck by drafting multiple tokens with a lightweight model and verifying them in parallel while preserving the target model's output distribution (Leviathan et al., 2023; Chen et al., 2023). Tree-based methods extend a single draft chain into multiple candidate continuations. In particular, context-aware dynamic trees deterministically select the tokens with the highest draft path scores to fill a candidate budget, so that the tree's depth and branching adapt to the current context (Li et al., 2024b; Ringel & Romano, 2026; Lin & Jang, 2026).

However, a stronger tree structure does not necessarily yield a stronger speculative decoder. Under stochastic decoding, the way candidates are generated and verified also matters. A sampled chain can be verified with Rejection Sampling or with Block Verification; the latter jointly verifies a draft block and is optimal in the expected number of tokens generated per iteration in single-chain speculative decoding (Sun et al., 2025). In our experiments, a deterministic dynamic tree outperforms the top-1 chain baseline, but falls behind a sampled chain with Block Verification on some tasks. Thus, comparing only against a deterministic chain can obscure an important limitation of dynamic trees.

This limitation stems from how dynamic trees select their candidates. Draft path scores guide the allocation of nodes to promising continuations, but local acceptance is then limited by the target probability mass of the retained high-score candidates. Random sampling, paired with appropriate verification, offers a different source of acceptance gains that deterministic selection does not exploit. Recent work has begun to introduce stochastic candidates within the standard dynamic-tree

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

construction process (Hu et al., 2026). We ask a more radical question: *can the planning of a tree's topology be fully separated from the sampling of its candidates?*

We answer this question with **Tsubame**, a two-pass tree speculative decoding framework that separates whole-tree topology planning from final candidate generation. The first pass uses draft path scores to construct and freeze a context-aware topology. The second pass then *replays* this frozen topology: it keeps the same structure and samples fresh tokens for every node, and these tokens, rather than the deterministic planning candidates, are submitted for target-model verification. We refer to this procedure as *tree replay*. This design retains the first pass's context-aware structural allocation while enabling random sampling and its associated verification rules.

With autoregressive drafters, replay would require rerunning a draft forward pass at every tree layer under the replayed tokens, effectively repeating the entire drafting stage. Diffusion-based drafters make this cost small: future-position distributions are either available from parallel prediction or can be obtained under new prefixes through lightweight conditional computation (Chen et al., 2026; Huang et al., 2026; Cheng et al., 2026; Inco AI, 2026), making full-tree replay practical. In our implementation, replay adds latency on the order of 1% of a decoding round for Domino and even less for DFlash2 (Appendix B.6).

We evaluate Tsubame with three diffusion-based drafters, Domino, DFlash2, and DSpark, on six datasets spanning math, code, and conversation. Paired with different sampling strategies and verifiers, replay can improve over the deterministic tree, with tree-level verifiers realizing these gains most consistently. In particular, with mixed sampling and UniVer, replay improves acceptance length and throughput over both the deterministic first-pass tree and sampled chains with Block Verification on all six datasets for Domino and DFlash2, reversing every task-level deficit of the deterministic tree against Block Verification. These gains persist as the candidate budget grows from 16 to 64, and in concurrent serving, replay with tree-level verification outperforms both the deterministic tree and Block Verification at every evaluated concurrency. Together, these results show that a well-planned topology and effective sampling and verification offer complementary, rather than competing, routes to higher acceptance and throughput.

Our contributions are threefold:

- We point out a largely overlooked gap in how dynamic trees are evaluated under stochastic decoding: comparisons against top-1 chains overstate the value of tree structure, as budget-matched sampled chains with advanced verification can outperform deterministic dynamic trees and thus serve as a more informative baseline.
- We propose Tsubame, a two-pass framework that freezes a context-aware topology in the first pass and replays it in the second, resampling every node for verification. We prove that Tsubame is lossless under any compatible sampling–verification pair.
- We show that Tsubame generalizes across diffusion-based drafters, tasks, and target models. Paired with suitable sampling and verification, it consistently improves acceptance length and throughput over deterministic trees, reverses their deficits against sampled chains, and retains its gains across candidate budgets and under concurrent serving.

2 Related Work and Background

2.1 Autoregressive and Diffusion-Based Drafters

Speculative decoding admits different forms of draft models. The original methods use a separate, smaller autoregressive language model to propose continuations for a larger target model (Leviathan et al., 2023; Chen et al., 2023). The EAGLE series instead trains a lightweight autoregressive draft model that takes target-model features as input (Li et al., 2024a; 2025). These methods allow the target model to verify multiple positions at once, but the drafter itself remains autoregressive: each draft forward pass advances the candidates by only one position.

Diffusion-based drafters address this remaining sequential bottleneck through parallel block prediction. DFlash produces distributions for multiple future positions in a single forward pass (Chen et al., 2026). Domino, DSpark, and DFlash2 further model within-block dependencies through lightweight mechanisms that avoid a full backbone pass for each draft position: Domino corrects logits with a

GRU state advanced along the prefix, DSpark applies a Markov-head correction conditioned on the preceding token, and DFlash2 uses precomputed transition scores between candidates (Huang et al., 2026; Cheng et al., 2026; Inco AI, 2026).

2.2 Tree-Structured Speculative Decoding

Tree speculative decoding proposes multiple continuation paths (Miao et al., 2024; Jeon et al., 2024; Xiong et al., 2024). Static methods use predefined or offline-optimized topologies, as in EAGLE and Sequoia (Li et al., 2024a; Chen et al., 2024). Context-aware methods such as EAGLE-2 and OPT-Tree instead use draft probabilities as a surrogate for acceptance, deterministically selecting candidates with high cumulative path probabilities to fill the verification budget (Li et al., 2024b; Wang et al., 2025), allowing the tree to adapt its depth and branching to the context.

Different drafters can also lead to different tree-construction strategies. Autoregressive methods typically expand trees layer by layer, as in EAGLE-2's expand-then-prune procedure (Li et al., 2024b). Diffusion-based drafters make node-wise expansion inexpensive: DDTree uses max-heap search over parallel draft distributions produced by DFlash, while DominoTree uses Domino as its drafter and performs the corresponding lightweight conditional updates during expansion without rerunning the full diffusion backbone (Ringel & Romano, 2026; Lin & Jang, 2026). Despite their different construction orders, these dynamic methods use draft path scores to select candidates deterministically and retain the selected tokens for verification.

2.3 Candidate Sampling and Verification

Under stochastic decoding ($T > 0$), acceptance depends on both candidate generation and verification. At a fixed prefix, let p and q denote the target and draft next-token distributions over vocabulary $\mathcal{V}$. Standard speculative decoding samples a candidate from q and applies rejection sampling (RS), accepting token v with probability $\min\{1, p(v)/q(v)\}$ and correcting the output distribution upon rejection (Leviathan et al., 2023; Chen et al., 2023). Its marginal acceptance probability is

$$\alpha_{\mathrm{RS}}(p, q) = \sum_{v \in \mathcal{V}} \min\{p(v), q(v)\}. \tag{1}$$

Alternatively, deterministically proposing $v^\star = \arg\max_v q(v)$ and matching it against a target sample gives acceptance probability $p(v^\star)$. Both approaches preserve the target distribution, but their acceptance probabilities differ. Li et al. (2026) report that RS often achieves higher acceptance rates than top-1 target-sample matching in native MTP models. Thus, even for a single draft token, how the candidate is generated and verified affects acceptance independently of any tree structure. With multiple candidates and positions, this design space expands to sampling schemes and multi-step verification rules, which we review in Appendix A.

2.4 Context-Aware Topology and Stochastic Proposals

Static tree designs, such as EAGLE-1, SpecInfer, and Sequoia, specify the topology in advance (Li et al., 2024a; Miao et al., 2024; Chen et al., 2024). In contrast, the context-aware constructions in EAGLE-2, OPT-Tree, DDTree, and DominoTree use candidate tokens and their draft path scores to adapt the topology (Li et al., 2024b; Wang et al., 2025; Ringel & Romano, 2026; Lin & Jang, 2026). As Table 1 shows, static trees leave candidate sampling and verification as a free choice, whereas dynamic trees largely do not. This is not a coincidence: introducing sampled candidates into a dynamic tree faces a substantive difficulty.

In a static tree, the topology is fixed before any token is drafted. Sampled tokens only fill predetermined slots, and their realizations affect neither the structure nor the survival of any predefined node, including their own. The candidates submitted for verification therefore follow the sampling distribution without distortion, and any compatible sampling–verification pair remains lossless. In a dynamic tree, the opposite holds: the topology is determined by the candidates themselves, as whether a node is expanded, retained, or pruned depends on its own draft path score. If a sampled token directly enters such a construction, its survival depends on its realized token, so the retained candidates no longer follow the sampling distribution, and a verifier that assumes this distribution loses its losslessness guarantee.

Table 1: Comparison of topology construction and sampling/verifier flexibility. Tsubame separates context-aware topology planning from the choice of candidate sampling and verification.

Tree	Method	Drafter	Construction	Context-aware	Flexible Samp. & Verif.*
Static	EAGLE-1	AR	Handcrafted	✗	✓
	SpecInfer	AR	Fixed branching	✗	✓
	Sequoia	AR	Offline DP	✗	✓
Dynamic	EAGLE-2/3	AR	Expand-then-prune	✓	✗
	RheoSampling	AR	EAGLE-2-style	✓	◐
	DDTree	Diffusion	Max-heap	✓	✗
	DominoTree	Diffusion+GRU	Max-heap	✓	✗
	Tsubame	Diffusion-based	Plan + Replay	✓	✓

*Flexibility denotes the ability to choose candidate sampling and proposal-aware verification separately from topology planning. For static trees, the mark denotes support in principle rather than implementation or evaluation in the original work. ◐ denotes RheoSampling's restricted support: one sampled candidate per local deterministic candidate group, placed according to prescribed rules.

RheoSampling addresses this issue within dynamic-tree construction by assigning each sampled candidate a prescribed proxy probability for structural scoring, separate from the actual sampling probability used for verification (Hu et al., 2026). This keeps the structural role of sampled candidates independent of their realization, but restricts sampling to one candidate per local deterministic group, placed according to prescribed rules.

Tsubame takes a more direct route: if a sampled token's realization must not influence which nodes survive, the structure can simply be fixed before any token is sampled. It plans the topology first and fills it afterwards, reducing candidate sampling and verification to the static-tree setting. As discussed in Section 1, diffusion-based drafters make this second pass practical.

3 Tsubame: Tree Planning and Replay

Tsubame first plans a context-aware tree topology, then replays the fixed topology to generate the candidates used for verification (Figure 1). This separation allows the planner and the sampling–verification scheme to be chosen independently.

3.1 First Pass: Tree Construction

The first pass allocates the candidate budget using draft path scores, adapting tree construction to how each drafter obtains its conditional distributions. For DFlash2, transition scores between candidates are precomputed, so conditional distributions can be obtained by table lookup, and we use node-wise max-heap expansion. Domino and DSpark instead require a lightweight correction step for each new prefix. For these drafters, we use EAGLE-2-style expand-then-prune, which batches corrections across frontier nodes rather than invoking them separately for each expanded node. In principle, Tsubame does not constrain how the first pass is built: any planner that produces a topology before replay can be used, chosen according to the properties of each drafter. Our choices favor efficiency; for example, batched expand-then-prune achieves higher throughput than node-wise max-heap expansion at larger budgets with similar acceptance length (Appendix B.3).

Branching increases the number of prefixes requiring conditional distributions, making full-vocabulary logit correction across branches expensive. We therefore restrict the draft vocabulary to a pool of top-scoring tokens from the backbone at each position (Appendix B). Logit corrections are computed only within this pool, and the corrected logits are normalized over the pool to obtain the draft distribution used for candidate selection.

The first pass produces a deterministic, context-aware dynamic tree, which also serves as our deterministic baseline (First) when verified directly. Tsubame discards the planning tokens and retains only the topology $\mathcal{T}$, consisting of the parent-child relations and child ordering.

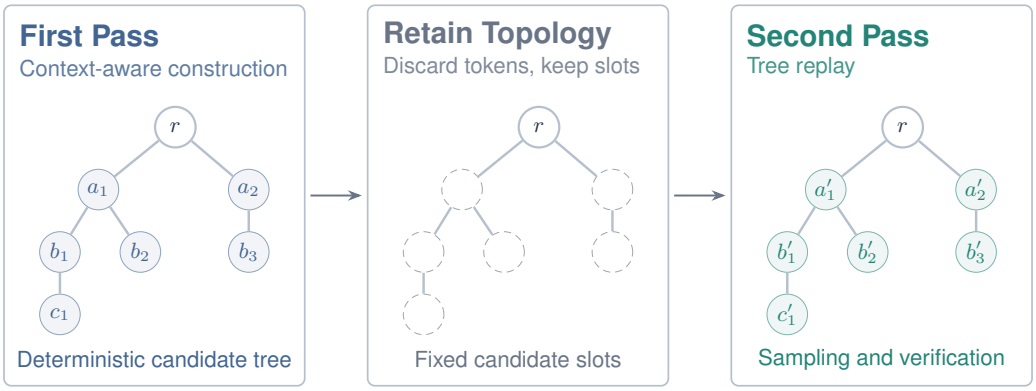


Figure 1: Overview of Tsubame. The first pass constructs a deterministic, context-aware candidate tree (left). Tsubame discards the planning tokens and retains the ordered topology (middle). The second pass regenerates candidates in the fixed slots using the chosen sampling scheme to form the tree used for target-model verification (right). Primed labels denote replayed candidates.

3.2 Second Pass: The "Kaeshi" Replay[1]

The second pass fills $\mathcal{T}$ layer by layer. Let x denote the current generation prefix and $y_{\mathrm{path}(u)}$ the replayed tokens along the path from the root to node u, excluding the root itself. This sequence is empty when u is the root. For each parent u, we compute

$$q_u(\cdot) = q\big(\cdot \mid x, y_{\mathrm{path}(u)}\big) \tag{2}$$

and populate its fixed child slots using the chosen sampling scheme. Each q_u is obtained by invoking the drafter's own conditional correction under the replayed prefix, restricted to the same sparse candidate pools as in the first pass, with parents at the same depth processed in parallel. The topology remains unchanged throughout replay.

Replay thus differs from the first pass only in candidate selection: it samples new candidates under the chosen scheme rather than retaining the planning tokens. We mainly use sampling without replacement (WoR) and mixed sampling, paired with the verifiers described in Appendix A.

3.3 Losslessness Guarantee

Tsubame inherits its losslessness from the static-tree verification algorithm used in the second pass. Appendix A describes the compatible sampling–verification pairs.

Proposition 1 (Losslessness of tree replay)**.** *Suppose that the first pass fixes the tree topology before replay and that the topology is not subsequently changed according to the replayed tokens. If the sampling–verification method used during replay is lossless for any fixed topology, then Tsubame preserves the target model's generation distribution.*

Proof. Fix the current decoding history and the topology produced by the first pass. Replay and verification then form a static-tree speculative decoding procedure on the fixed topology. Its output therefore follows the target model's distribution by the losslessness of the chosen verifier. Since this holds for every topology produced by the first pass, marginalizing over the topology and repeating the argument across decoding rounds preserves the target sequence distribution. □

In particular, Mixed/UniVer is lossless because UniVer provides this guarantee for any fixed tree topology under its compatible sampling rule (Weng et al., 2026). During verification, however, the probabilities must correspond to the candidates generated during replay, not to the planning tokens discarded after the first pass. If replay uses a restricted candidate pool, these probabilities are normalized within that pool.

[1]The name alludes to *Tsubame-gaeshi*. Here it evokes a second pass over the planned tree.

Table 2: Guide to the verifiers used in our experiments. Scope distinguishes step-wise from sequence-level verification. The sampling column lists the configurations evaluated here, rather than all supported schemes. All pairs preserve the target distribution.

Structure	Verifier	Scope	Sampling	Description
Chain	Naive	-	Top1	Target-sample matching
	RS	Step-wise	Random	Standard rejection sampling
	BlockV	Sequence-level	Random	Improves RS through sequence-level coupling
Tree	Naive	-	TopK	Target-sample matching
	RRS	Step-wise	WoR	Standard recursive rejection across siblings
	Traversal	Sequence-level	WoR	RRS with sequence-level coupling
	UniVer	Sequence-level	Mixed	Unified verification supporting flexible sampling strategies

4 EXPERIMENTS

We evaluate whether replay improves acceptance and throughput over the deterministic first-pass tree, whether it outperforms sampled chains equipped with advanced verification, and how these effects vary with the candidate budget, the sampling–verification pair, and concurrent serving.

4.1 EXPERIMENTAL SETUP

Models and datasets. Our main experiments use DFlash2 (Inco AI, 2026) and Domino (Huang et al., 2026), with DSpark (Cheng et al., 2026) results provided in Appendix B.4. Domino and DSpark use Qwen3-8B (Yang et al., 2025) as the default target model with thinking disabled, while DFlash2 uses Muse Glimmer 30B (Meta Superintelligence Lab, 2026). Additional DFlash2 results with Qwen3.8-27B (Qwen Team, 2026) are reported in Appendix B.4. We evaluate on six datasets spanning three tasks: math (GSM8K (Cobbe et al., 2021) and Math500 (Lightman et al., 2023)), code (HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021)), and conversation (MT-Bench (Zheng et al., 2023) and Alpaca (Taori et al., 2023)).

Proposal and verification strategies. For chains, we compare deterministic top-1 proposals with target-sample matching (Top1), and sampled proposals with rejection sampling (RS) (Leviathan et al., 2023; Chen et al., 2023) or Block Verification (Sun et al., 2025). For trees, we mainly use two sampling schemes: standard sampling without replacement (WoR), paired with layer-wise RRS or Traversal Verification (Chen et al., 2024; Weng et al., 2025); and mixed sampling (Hu et al., 2025), paired with UniVer (Weng et al., 2026). Table 2 summarizes their differences; please refer to Appendix A for detailed discussion on sampling and verification strategies.

Metrics and implementations. We report speedup ratio (SR) relative to autoregressive decoding and average acceptance length (τ), with calculation details in Appendix B. Unless otherwise specified, we use temperature $T = 1$, a maximum generation length of 512 tokens, and a candidate budget of $B = 16$ nodes including the root, which also sets the draft length for chains. Single-request Qwen3-8B experiments run on NVIDIA GeForce RTX 3090 GPUs, while serving experiments and all Muse Glimmer 30B and Qwen3.8-27B experiments use NVIDIA RTX PRO 6000 GPUs with 96 GB of memory. Unless otherwise noted, we pool results over three random seeds. All tree-construction and verification implementations use CUDA Graphs.

4.2 MAIN RESULTS

Tables 3 and 4 distinguish the benefit of tree structure from the effects of candidate generation and verification at the same budget, $B = 16$. For both drafters, deterministic dynamic trees achieve longer acceptance lengths than Top1 chains on every dataset, showing a clear structural advantage over a single deterministic continuation.

However, this advantage shrinks when chains use random sampling with RS or Block Verification. DFlash2 exhibits the clearest reversal: Block Verification exceeds the deterministic tree in throughput on five of the six datasets, with MT-Bench essentially tied, and in acceptance length on four.

Table 3: DFlash2 with Muse Glimmer 30B results on all six datasets at $B = 16$. TopK denotes deterministic candidate selection in the first pass, and Naive denotes target-sample matching. Bold marks the best value in each column. The underlined verifier denotes the default configuration adopted by DFlash2.

			Math				Code				Conversation			
			GSM8K		Math500		HumanEval		MBPP		MT-Bench		Alpaca	
Structure	Sampling	Verifier	τ	SR	τ	SR	τ	SR	τ	SR	τ	SR	τ	SR
Chain	Top1	Naive	5.44	4.15×	5.42	4.14×	4.41	3.35×	3.75	2.89×	3.38	2.55×	2.97	2.29×
	Random	<u>RS</u>	6.25	4.67×	6.12	4.58×	5.48	4.08×	4.82	3.63×	3.94	2.91×	3.62	2.73×
	Random	BlockV	6.52	4.87×	6.25	4.67×	5.67	4.22×	5.07	3.82×	4.12	3.04×	3.70	2.79×
Tree (First)	TopK	Naive	6.39	4.75×	6.21	4.62×	5.42	4.02×	4.69	3.52×	4.14	3.04×	3.70	2.78×
Tree (Replay)	WoR	RRS	6.67	4.94×	6.59	4.89×	5.85	4.32×	5.12	3.83×	4.39	3.21×	3.96	2.97×
	WoR	Traversal	**6.83**	**5.06**×	6.68	4.96×	**5.97**	**4.42**×	**5.30**	**3.97**×	**4.50**	**3.30**×	**4.03**	**3.02**×
	Mixed	UniVer	6.79	5.03×	**6.70**	**4.97**×	5.96	4.40×	5.25	3.92×	4.46	3.26×	4.02	3.01×

Table 4: Domino with Qwen3-8B results on all six datasets at $B = 16$. All methods use sparse64 draft candidate pools. TopK denotes deterministic candidate selection in the first pass, and Naive denotes target-sample matching. Bold marks the best value in each column. The underlined verifier denotes the default configuration adopted by Domino.

			Math				Code				Conversation			
			GSM8K		Math500		HumanEval		MBPP		MT-Bench		Alpaca	
Structure	Sampling	Verifier	τ	SR	τ	SR	τ	SR	τ	SR	τ	SR	τ	SR
Chain	Top1	<u>Naive</u>	7.21	5.54×	5.96	4.62×	5.49	4.26×	5.16	4.08×	3.14	2.44×	2.87	2.22×
	Random	RS	8.05	6.14×	6.58	5.06×	6.08	4.68×	5.66	4.45×	3.56	2.75×	3.20	2.47×
	Random	BlockV	8.38	6.38×	6.98	5.35×	6.30	4.83×	5.87	4.60×	3.73	2.87×	3.35	2.57×
Tree (First)	TopK	Naive	8.00	6.04×	6.75	5.16×	6.49	4.97×	6.30	4.93×	4.26	3.24×	3.97	3.04×
Tree (Replay)	WoR	RRS	8.37	6.31×	7.19	5.46×	6.57	4.98×	6.23	4.84×	4.22	3.19×	3.85	2.93×
	WoR	Traversal	**8.59**	**6.45**×	7.23	5.49×	**6.88**	**5.21**×	6.44	4.99×	4.35	3.29×	3.96	3.01×
	Mixed	UniVer	8.57	6.44×	**7.26**	**5.51**×	6.87	5.19×	**6.48**	**5.02**×	**4.46**	**3.36**×	**4.10**	**3.11**×

Domino shows the same limitation on the two math tasks and, more mildly, on HumanEval: on HumanEval, the deterministic tree's acceptance advantage narrows from 6.49 versus 5.49 against Top1 to 6.49 versus 6.30 against Block Verification, while on GSM8K and Math500 the ordering reverses. Comparing only against Top1 thus overstates the advantage of deterministic trees; Block Verification, which is optimal in single-chain speculative decoding, provides a more informative reference.

These comparisons suggest complementary roles: deterministic trees improve structural allocation, while stochastic candidates paired with suitable verification offer additional acceptance gains. Replay combines the two by retaining the planned topology while regenerating its candidates. With Mixed/UniVer, replay improves both acceptance length and throughput over the first pass and Block Verification on all six datasets for both drafters, reversing every deficit of the deterministic tree against Block Verification; on Domino, for example, the throughput deficits on GSM8K and Math500 become gains of 1.00% and 2.93%. The pooled throughput gains over the first pass are 8.51% for DFlash2 and 4.19% for Domino, showing that the additional acceptance more than offsets replay's extra computation. WoR/Traversal achieves comparable overall performance.

4.3 Budget Scaling and Verification Choices

Replay vs. deterministic drafting. Increasing the candidate budget strengthens the deterministic tree but does not remove the benefit of replay. Tables 5 and 6 show that the first pass achieves longer acceptance lengths as the budget grows from 16 to 64, while Mixed/UniVer improves both acceptance length and throughput on every dataset at each budget for both drafters. Its relative

Table 5: DFlash2 with Muse Glimmer 30B across candidate budgets on all six datasets. Dataset cells report τ / SR. Replay uses WoR/RRS, WoR/Traversal, or Mixed/UniVer. The final column reports the respective changes relative to First pass at the same budget. Bold marks the best value for each metric within each budget. The percentages are calculated using the values before rounding.

		Math		Code		Conversation		
Budget	Method	GSM8K	Math500	HumanEval	MBPP	MT-Bench	Alpaca	Gain τ / SR
16	First	6.39 / 4.75×	6.21 / 4.62×	5.42 / 4.02×	4.69 / 3.52×	4.14 / 3.04×	3.70 / 2.78×	–
	Replay (RRS)	6.67 / 4.94×	6.59 / 4.89×	5.85 / 4.32×	5.12 / 3.83×	4.39 / 3.21×	3.96 / 2.97×	+6.97% / +6.73%
	Replay (Traversal)	**6.83 / 5.06×**	6.68 / 4.96×	**5.97 / 4.42×**	**5.30 / 3.97×**	**4.50 / 3.30×**	**4.03 / 3.02×**	**+9.45% / +9.20%**
	Replay (UniVer)	6.79 / 5.03×	**6.70 / 4.97×**	5.96 / 4.40×	5.25 / 3.92×	4.46 / 3.26×	4.02 / 3.01×	+8.88% / +8.51%
32	First	7.07 / 4.96×	7.04 / 4.94×	6.00 / 4.20×	5.28 / 3.74×	4.60 / 3.18×	4.09 / 2.90×	–
	Replay (RRS)	7.33 / 5.14×	7.20 / 5.05×	6.36 / 4.44×	5.66 / 4.00×	4.74 / 3.28×	4.27 / 3.03×	+4.50% / +4.41%
	Replay (Traversal)	7.49 / 5.25×	7.38 / 5.18×	6.52 / 4.56×	**5.82 / 4.11×**	**4.92 / 3.40×**	**4.42 / 3.13×**	+7.67% / +7.53%
	Replay (UniVer)	**7.54 / 5.28×**	**7.52 / 5.26×**	**6.57 / 4.58×**	5.80 / 4.10×	4.91 / 3.39×	**4.42 / 3.13×**	**+7.95% / +7.68%**
64	First	7.77 / 5.32×	7.67 / 5.26×	6.56 / 4.48×	5.75 / 3.97×	4.94 / 3.34×	4.42 / 3.06×	–
	Replay (RRS)	7.86 / 5.38×	7.69 / 5.27×	6.85 / 4.67×	6.02 / 4.16×	5.06 / 3.42×	4.56 / 3.15×	+2.93% / +2.81%
	Replay (Traversal)	7.90 / 5.41×	7.81 / 5.35×	6.94 / 4.74×	6.22 / 4.30×	5.12 / 3.46×	4.64 / 3.22×	+4.72% / +4.67%
	Replay (UniVer)	**8.14 / 5.58×**	**8.06 / 5.53×**	**6.95 / 4.75×**	**6.23 / 4.31×**	**5.26 / 3.56×**	**4.71 / 3.27×**	**+6.36% / +6.48%**

Table 6: Domino with Qwen3-8B across candidate budgets on all six datasets. Dataset cells report τ / SR. Replay uses WoR/RRS, WoR/Traversal, or Mixed/UniVer. The final column reports the respective τ / SR changes relative to First pass at the same budget. Bold marks the best value for each metric within each budget. The percentages are calculated using the values before rounding.

		Math		Code		Conversation		
Budget	Method	GSM8K	Math500	HumanEval	MBPP	MT-Bench	Alpaca	Gain τ / SR
16	First	8.00 / 6.04×	6.75 / 5.16×	6.49 / 4.97×	6.30 / 4.93×	4.26 / 3.24×	3.97 / 3.04×	–
	Replay (RRS)	8.37 / 6.31×	7.19 / 5.46×	6.57 / 4.98×	6.23 / 4.84×	4.22 / 3.19×	3.85 / 2.93×	+0.71% / +0.09%
	Replay (Traversal)	**8.59 / 6.45×**	7.23 / 5.49×	**6.88 / 5.21×**	6.44 / 4.99×	4.35 / 3.29×	3.96 / 3.01×	+3.75% / +2.98%
	Replay (UniVer)	8.57 / 6.44×	**7.26 / 5.51×**	6.87 / 5.19×	**6.48 / 5.02×**	**4.46 / 3.36×**	**4.10 / 3.11×**	**+5.10% / +4.19%**
32	First	9.36 / 6.18×	7.79 / 5.22×	7.35 / 4.93×	7.30 / 5.00×	4.78 / 3.20×	4.43 / 2.97×	–
	Replay (RRS)	9.53 / 6.28×	7.88 / 5.24×	7.44 / 4.95×	7.11 / 4.83×	4.69 / 3.12×	4.29 / 2.85×	-0.74% / -1.44%
	Replay (Traversal)	**9.72 / 6.39×**	8.16 / 5.41×	7.67 / 5.10×	7.24 / 4.91×	4.83 / 3.20×	4.38 / 2.91×	+1.97% / +1.10%
	Replay (UniVer)	9.70 / 6.33×	**8.21 / 5.44×**	**7.80 / 5.17×**	**7.59 / 5.15×**	**5.01 / 3.32×**	**4.62 / 3.06×**	**+4.90% / +3.80%**
64	First	10.32 / 6.81×	8.46 / 5.67×	8.14 / 5.45×	8.05 / 5.51×	5.21 / 3.48×	4.84 / 3.25×	–
	Replay (RRS)	10.16 / 6.69×	8.46 / 5.62×	8.04 / 5.35×	7.73 / 5.26×	5.08 / 3.37×	4.63 / 3.08×	-2.21% / -2.84%
	Replay (Traversal)	10.39 / 6.83×	8.79 / 5.83×	8.31 / 5.51×	8.06 / 5.47×	5.23 / 3.46×	4.79 / 3.18×	+0.99% / +0.03%
	Replay (UniVer)	**10.62 / 6.93×**	**8.85 / 5.87×**	**8.58 / 5.69×**	**8.29 / 5.62×**	**5.44 / 3.60×**	**5.11 / 3.39×**	**+4.60% / +3.53%**

gains narrow as the deterministic baseline becomes stronger, yet at $B = 64$ it still improves pooled throughput by 6.48% for DFlash2 and 3.53% for Domino. Allocating more nodes and improving how those nodes are populated and verified are therefore complementary.

Advanced vs. vanilla verification. The comparison between layer-wise RRS and Traversal clarifies the role of verification. Both use WoR candidates, but Traversal achieves higher pooled acceptance lengths and throughput at every budget for both drafters. In DFlash2, both methods' relative gains narrow as the budget grows, but Traversal retains a larger fraction of its $B = 16$ acceptance gain at $B = 64$. In Domino, the RRS acceptance gain becomes negative as the budget increases, whereas Traversal's remains positive. Resampling candidates alone thus does not fully exploit the multi-step continuations in the planned tree; a verifier that accounts for sequence-level dependencies makes more effective use of the same budget.

Impact of sampling strategy. Under WoR sampling, UniVer is mathematically equivalent to Traversal Verification (Weng et al., 2026), so WoR/Traversal and Mixed/UniVer compare sampling strategies within the same verification framework. For DFlash2, WoR/Traversal is slightly stronger at $B = 16$, while Mixed/UniVer is strongest at $B = 32$ and $B = 64$. For Domino, Mixed/UniVer achieves the highest pooled acceptance length and throughput at all three budgets, with its acceptance advantage over WoR/Traversal widening as the budget grows. These results motivate choosing the sampling strategy according to the task, model, and candidate budget (see also Appendix B.4).

Table 7: DFlash2 with Muse Glimmer serving throughput (TPS) on a single RTX PRO 6000 GPU at $B = 16$, pooled over GSM8K, HumanEval, and MT-Bench with three seeds. C denotes concurrency, and parenthesized percentages are relative to the RS chain. The underlined verifier denotes the default chain verifier adopted by DFlash2. Bold marks the highest throughput in each column.

Structure	Sampling	Verifier	$C = 1$	$C = 2$	$C = 4$	$C = 8$	$C = 16$	$C = 32$
Chain	Random	<u>RS</u>	114.7	216.3	418.8	723.8	1153.6	1449.8
	Random	BlockV	118.9 (+3.6%)	224.6 (+3.9%)	434.2 (+3.7%)	744.9 (+2.9%)	1189.4 (+3.1%)	1504.5 (+3.8%)
Tree (First)	TopK	Naive	112.9 (-1.5%)	215.1 (-0.5%)	414.7 (-1.0%)	711.8 (-1.7%)	1151.1 (-0.2%)	1451.2 (+0.1%)
Tree (Replay)	WoR	RRS	120.7 (+5.2%)	227.1 (+5.0%)	439.4 (+4.9%)	755.3 (+4.3%)	1208.3 (+4.7%)	1527.5 (+5.4%)
	WoR	Traversal	**123.1 (+7.4%)**	**232.4 (+7.5%)**	**449.0 (+7.2%)**	**771.6 (+6.6%)**	1230.2 (+6.6%)	**1569.7 (+8.3%)**
	Mixed	UniVer	122.2 (+6.6%)	231.7 (+7.1%)	446.2 (+6.5%)	767.7 (+6.1%)	**1237.1 (+7.2%)**	1556.4 (+7.4%)

Table 8: Domino with Qwen3-8B serving throughput (TPS) on a single RTX PRO 6000 GPU at $B = 16$, pooled over GSM8K, HumanEval, and MT-Bench with three seeds. C denotes concurrency, and parenthesized percentages are relative to the Top1 chain. The underlined verifier denotes the default chain verifier adopted by Domino. Bold marks the highest throughput in each column.

Structure	Sampling	Verifier	$C = 2$	$C = 4$	$C = 8$	$C = 16$	$C = 32$
Chain	Top1	<u>Naive</u>	658.8	1253.2	2232.1	3407.3	4301.8
	Random	RS	684.4 (+3.9%)	1275.5 (+1.8%)	2281.7 (+2.2%)	3530.4 (+3.6%)	4416.5 (+2.7%)
	Random	BlockV	701.4 (+6.5%)	1322.7 (+5.5%)	2365.8 (+6.0%)	3629.0 (+6.5%)	4542.1 (+5.6%)
Tree (First)	TopK	Naive	710.3 (+7.8%)	1326.6 (+5.9%)	2352.3 (+5.4%)	3636.9 (+6.7%)	4544.0 (+5.6%)
Tree (Replay)	WoR	RRS	700.5 (+6.3%)	1311.2 (+4.6%)	2368.0 (+6.1%)	3606.1 (+5.8%)	4544.5 (+5.6%)
	WoR	Traversal	722.2 (+9.6%)	1346.9 (+7.5%)	2410.5 (+8.0%)	3677.9 (+7.9%)	4667.2 (+8.5%)
	Mixed	UniVer	**728.4 (+10.6%)**	**1357.1 (+8.3%)**	**2419.0 (+8.4%)**	**3709.2 (+8.9%)**	**4677.1 (+8.7%)**

4.4 Serving Throughput

Concurrent serving tests whether replay remains beneficial when multiple requests share execution resources. We evaluate both drafters at $B = 16$ on GSM8K, HumanEval, and MT-Bench at concurrencies up to 32 (Tables 7 and 8). For DFlash2, the deterministic tree stays close to the RS chain and below Block Verification at every concurrency, so structural allocation alone yields no throughput advantage. For Domino, the deterministic tree is already competitive with Block Verification, trailing it only at $C = 8$.

In both settings, replay with Traversal or UniVer outperforms the deterministic tree and Block Verification at every concurrency. Relative to Block Verification, WoR/Traversal and Mixed/UniVer improve throughput by 3.4%–4.3% and 2.8%–4.0% for DFlash2, and by 1.3%–3.0% and 2.2%–3.8% for Domino. Replay's acceptance gains thus continue to offset its additional computation under concurrent serving, including when the deterministic tree is already competitive.

5 Conclusion

This work examines the interplay between tree structure, candidate sampling, and verification in speculative decoding. Our analysis shows that the structural advantage of deterministic dynamic trees does not always compensate for the gains available to sampled chains with advanced verification. We introduce Tsubame to combine these complementary benefits: it first plans a context-aware topology, then replays that topology to generate stochastic candidates for verification. Diffusion-based drafters make this second pass inexpensive, while compatible verification guarantees that the target distribution is preserved. Experiments across DFlash2, Domino, and DSpark demonstrate improvements in acceptance length and throughput, including cases where replay reverses the disadvantage of deterministic trees against advanced chain baselines. These benefits extend to larger candidate budgets and concurrent serving. Together, our findings show that topology planning need not determine the final candidates: separating these decisions provides a practical way to combine structural allocation with effective sampling and verification.

AI USE STATEMENT

Tasks requiring disclosure. (i) *Proofs.* The losslessness argument (Proposition 1) was developed by the authors; AI tools helped write out the proof. No mathematical claim in this paper was formulated by AI tools. (ii) *Implementation.* AI tools assisted in writing code for tree construction, replay, verification, and evaluation; all code was verified by the authors. AI tools were also used to audit the implementation and run diagnostic tests. (iii) *Experimental design and interpretation.* The research question, method, and experimental protocol were designed by the authors. AI tools provided feedback on baseline selection and result presentation, and helped check that textual claims match the reported numbers. (iv) *Translation.* AI tools were used to translate between the authors' working language and English when drafting and reviewing the manuscript. Synthetic data generation, dataset cleaning or reformatting, and qualitative data analysis do not apply to this work.

Other uses. AI tools were also used to draft and edit parts of the manuscript from the authors' outlines, to suggest the organization of sections and tables, and to search for and summarize related work.

The authors reviewed all AI-assisted text, proofs, and code, and take full responsibility for the content of this paper.

REPRODUCIBILITY STATEMENT

Section 3 describes the method, and Section 4.1 and Appendix B specify all evaluation settings, including sample counts, candidate pools, aggregation rules, seeds, and hardware. All drafters, target models, and datasets are publicly available (Appendix D).

REFERENCES

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

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

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026. URL https://arxiv.org/abs/2602.06036.

Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2402.12374.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, Qinyu Chen, Chengqi Deng, Jiping Yu, Damai Dai, Zhengyan Zhang, Yixuan Wei, Yixuan Tan, Wenkai Yang, Runxin Xu, Yu Wu, Zhean Xu, Xuanyu Wang, Muyang Chen, Rui Tian, Xiao Bi, Zhewen Hao, Shaoyuan Chen, Huanqi Cao, Wentao Zhang, Anyi Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026. URL https://arxiv.org/abs/2607.05147.

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

Qiao Hu, Yepeng Weng, Bo Zhang, and Takehisa Yairi. RheoSampling: Resolving the one-hot dilemma in stochastic dynamic-tree speculative decoding. *arXiv preprint arXiv:2609.21827*, 2026. URL https://arxiv.org/abs/2609.21827.

Zhengmian Hu, Tong Zheng, Vignesh Viswanathan, Ziyi Chen, Ryan A. Rossi, Yihan Wu, Dinesh Manocha, and Heng Huang. Towards optimal multi-draft speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2502.18779.

Jianuo Huang, Yaojie Zhang, Qituan Zhang, Hao Lin, Hanlin Xu, and Linfeng Zhang. Domino: Decoupling causal modeling from autoregressive drafting in speculative decoding. *arXiv preprint arXiv:2605.29707*, 2026. URL https://arxiv.org/abs/2605.29707.

Inco AI. DFlash 2: Keep Drafting Parallel, August 2026. URL https://inco.ai/blog/dflash2/.

Wonseok Jeon, Mukul Gagrani, Raghavv Goel, Junyoung Park, Mingu Lee, and Christopher Lott. Recursive speculative decoding: Accelerating LLM inference via sampling without replacement. *arXiv preprint arXiv:2402.14160*, 2024. URL https://arxiv.org/abs/2402.14160.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yucheng Li, Huiqiang Jiang, Yang Xu, Jianxin Yang, Yi Zhang, Yizhong Cao, Yuhao Shen, Fan Zhou, Rui Men, Jianwei Zhang, An Yang, Bowen Yu, Bo Zheng, Fei Huang, Junyang Lin, Dayiheng Liu, and Jingren Zhou. Breaking entropy bounds: Accelerating RL training via MTP with rejection sampling. *arXiv preprint arXiv:2606.12370*, 2026. URL https://arxiv.org/abs/2606.12370.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948. PMLR, 2024a. URL https://proceedings.mlr.press/v235/li24bt.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432. Association for Computational Linguistics, 2024b. URL https://aclanthology.org/2024.emnlp-main.422/.

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

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. URL https://arxiv.org/abs/2305.20050.

Saw S. Lin and Jyh-Shing Roger Jang. DominoTree: Conditional tree-structured drafting with Domino for speculative decoding. *arXiv preprint arXiv:2607.08642*, 2026. URL https://arxiv.org/abs/2607.08642.

Meta Superintelligence Lab. Muse glimmer model card. Hugging Face model card, 2026. URL https://huggingface.co/meta-models/Muse-Glimmer-30B.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating 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*, 2024. doi: 10.1145/3620666.3651335. URL https://arxiv.org/abs/2305.09781.

Qwen Team. Qwen3.8-27B model card. Hugging Face model card, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B.

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees. *arXiv preprint arXiv:2604.12989*, 2026. URL https://arxiv.org/abs/2604.12989.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. SpecTr: Fast speculative decoding via optimal transport. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2310.15141.

Ziteng Sun, Uri Mendlovic, Yaniv Leviathan, Asaf Aharoni, Jae Hun Ro, Ahmad Beirami, and Ananda Theertha Suresh. Block verification accelerates speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=frsg32u0rO.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following LLaMA model. GitHub repository, 2023. URL https://github.com/tatsu-lab/stanford_alpaca.

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. doi: 10.1162/tacl_a_00735. URL https://aclanthology.org/2025.tacl-1.8/.

Yepeng Weng, Qiao Hu, Xujie Chen, Li Liu, Dianwen Mei, Huishi Qiu, Jiang Tian, and Zhongchao Shi. Traversal verification for speculative tree decoding. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2505.12398.

Yepeng Weng, Qiao Hu, and Takehisa Yairi. UniVer: A unified perspective for multi-step and multi-draft speculative decoding. *arXiv preprint arXiv:2605.04543*, 2026. URL https://arxiv.org/abs/2605.04543.

Yunfan Xiong, Ruoyu Zhang, Yanzeng Li, Tianhao Wu, and Lei Zou. DySpec: Faster speculative decoding with dynamic token tree structure. *arXiv preprint arXiv:2410.11744*, 2024. URL https://arxiv.org/abs/2410.11744.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2306.05685.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2312.07104.

A Additional Background on Sampling and Verification

Section 2.3 contrasts RS with top-1 target-sample matching. More generally, target-sample matching accepts a child from a fixed candidate set C with probability $\sum_{v \in C} p(v)$. With multiple candidates and positions, both the sampling scheme and the verification rule provide further design choices.

With multiple candidates at one position, the sampling scheme is a separate design choice. Sampling with replacement draws independently from q, allowing repeated tokens. Sampling without replacement (WoR) excludes previously drawn tokens and renormalizes q before each subsequent draw. Mixed sampling, corresponding to the greedy draft sampling scheme of Hu et al. (2025), selects all but one candidate deterministically by draft probability and samples the last from the

renormalized remainder. These schemes induce different joint candidate distributions and can have different optimal acceptance rates (Hu et al., 2025).

Each sampling scheme must be paired with a compatible verifier. With-replacement candidates can be verified using Recursive Rejection Sampling (RRS), as in SpecInfer (Miao et al., 2024), or alternative rules such as K-Seq from SpecTr (Sun et al., 2023). WoR candidates are commonly paired with WoR variants of RRS, as in Sequoia, EAGLE, and Recursive Speculative Decoding (RSD) (Chen et al., 2024; Li et al., 2024a; Jeon et al., 2024). Even under the same candidate distribution, different verifiers can achieve different acceptance rates while all preserving the target distribution. For fixed target and draft distributions and a candidate budget, the sampling strategy determines the attainable acceptance ceiling, while the verifier determines how closely it is approached (Sun et al., 2023; Hu et al., 2025).

Beyond a single token or multiple candidates at one position, chains and trees introduce multi-step dependencies that verification must also account for. Block Verification shows that token-wise RS need not maximize expected token yield over multiple positions and uses sequence-level probabilities to make acceptance decisions, but is restricted to chains (Sun et al., 2025). Traversal Verification extends sequence-level verification to trees through sequence-level RRS with sampling without replacement (Weng et al., 2025). UniVer accommodates different sampling strategies within a unified verification framework and reduces exactly to Traversal Verification under WoR sampling (Weng et al., 2026).

B Additional Experimental Details and Results

B.1 Evaluation Samples and Aggregation

Evaluation samples. We follow Domino's per-dataset sample counts (Huang et al., 2026), evaluating on six datasets: 128 GSM8K, 128 Math500, 164 HumanEval, 128 MBPP, 80 MT-Bench, and 128 Alpaca samples. A multiturn prompt is treated as one sample. Serving experiments use GSM8K, HumanEval, and MT-Bench with 3 seeds, generating 512 tokens per request.

Draft candidate pools. DFlash2 natively uses 16-token candidate pools, which we retain in the Muse experiments. For Domino, we introduce 64-token pools for both chains and trees; Appendix B.7 provides the corresponding full-vocabulary control. For DSpark, we introduce sparse candidate pools for trees using the dual-1024 configuration, while its native chain remains dense. These restrictions apply only to draft proposals, not to the target distribution.

Pooled and macro aggregation. Unless otherwise specified, all reported metrics use pooled aggregation across the samples and seeds in each comparison group. Throughput is total output tokens divided by total recorded time. Speedup ratio (SR) is the ratio of speculative to AR throughput for the same target model, dataset, hardware, and backend. Let A_i and R_i denote the sum of recorded acceptance lengths and the number of decoding rounds for sample i, treating each seed's evaluation as a separate observation. Pooled and sample-level macro acceptance lengths are defined as

$$\tau_{\text{pooled}} = \frac{\sum_{i=1}^{N} A_i}{\sum_{i=1}^{N} R_i}, \qquad \tau_{\text{macro}} = \frac{1}{N} \sum_{i=1}^{N} \frac{A_i}{R_i}. \tag{3}$$

Pooled aggregation weights each sample by its number of decoding rounds, whereas macro aggregation gives each sample equal weight. The two statistics can therefore differ when samples have different round counts and acceptance lengths. Cross-dataset pooled results combine the underlying totals across datasets using the same rule.

Table 9 compares both acceptance-length statistics separately for each dataset and drafter. Within each dataset, macro values sum the per-sample means across the three seeds and divide by the number of sample–seed observations. This breakdown shows how the choice of aggregation affects the reported acceptance length on individual datasets.

Table 9: Pooled and sample-level macro acceptance lengths for each dataset, combining three seeds for the methods in Tables 3 and 4. Macro gives each sample–seed observation equal weight.

	GSM8K		Math500		HumanEval		MBPP		MT-Bench		Alpaca	
Method	Pooled	Macro	Pooled	Macro	Pooled	Macro	Pooled	Macro	Pooled	Macro	Pooled	Macro
Domino with Qwen3-8B												
Top1	7.21	7.88	5.96	6.78	5.49	5.65	5.16	5.64	3.14	4.05	2.87	3.20
RS	8.05	8.55	6.58	7.32	6.08	6.27	5.66	6.06	3.56	4.36	3.20	3.43
BlockV	8.38	8.88	6.98	7.71	6.30	6.48	5.87	6.32	3.73	4.55	3.35	3.55
First	8.00	8.48	6.75	7.45	6.49	6.66	6.30	6.68	4.26	5.00	3.97	4.22
WoR/RRS	8.37	8.83	7.19	7.80	6.57	6.73	6.23	6.64	4.22	4.92	3.85	4.01
WoR/Traversal	8.59	9.03	7.23	7.83	6.88	7.04	6.44	6.85	4.35	5.12	3.96	4.12
Mixed/UniVer	8.57	9.09	7.26	7.84	6.87	7.06	6.48	6.86	4.46	5.14	4.10	4.27
DFlash2 with Muse Glimmer 30B												
Top1	5.44	5.83	5.42	5.82	4.41	4.55	3.75	3.82	3.38	3.69	2.97	3.18
RS	6.25	6.65	6.12	6.55	5.48	5.64	4.82	4.91	3.94	4.30	3.62	3.88
BlockV	6.52	6.88	6.25	6.71	5.67	5.83	5.07	5.17	4.12	4.47	3.70	4.01
First	6.39	6.74	6.21	6.58	5.42	5.54	4.69	4.76	4.14	4.44	3.70	3.94
WoR/RRS	6.67	6.96	6.59	6.95	5.85	5.96	5.12	5.19	4.39	4.70	3.96	4.19
WoR/Traversal	6.83	7.14	6.68	7.05	5.97	6.11	5.30	5.38	4.50	4.83	4.03	4.30
Mixed/UniVer	6.79	7.10	6.70	7.06	5.96	6.08	5.25	5.32	4.46	4.78	4.02	4.25

B.2 Six-Dataset Aggregates and Cross-Seed Variability

We aggregate the six-dataset results from Tables 3 and 4 at $B = 16$. For each method and seed, we pool the recorded totals across all six datasets to obtain acceptance length and throughput. SR is the ratio of pooled speculative to pooled AR throughput for the same six-dataset workload, with the AR reference held fixed across seeds. Table 10 reports the all-seed pooled results alongside the mean and sample standard deviation (SD) of the three seed-level estimates.

Table 10: Six-dataset aggregates at $B = 16$ for the methods in the two main tables. Pooled combines all datasets and seeds; seed mean $\pm$ SD and 95% CIs use the three separately pooled seed-level estimates. SR is expressed as a multiplicative speedup.

	τ			SR ($\times$)		
Method	Pooled	Seed mean $\pm$ SD	95% CI	Pooled	Seed mean $\pm$ SD	95% CI
Domino with Qwen3-8B						
Top1	4.539	4.539 ± 0.009	[4.516, 4.563]	3.524	3.524 ± 0.007	[3.506, 3.542]
RS	5.065	5.065 ± 0.013	[5.032, 5.098]	3.904	3.904 ± 0.009	[3.882, 3.927]
BlockV	5.291	5.291 ± 0.018	[5.245, 5.336]	4.066	4.066 ± 0.013	[4.032, 4.100]
First	5.676	5.676 ± 0.015	[5.639, 5.714]	4.339	4.339 ± 0.016	[4.299, 4.380]
WoR/RRS	5.716	5.716 ± 0.016	[5.676, 5.757]	4.343	4.343 ± 0.012	[4.313, 4.373]
WoR/Traversal	5.889	5.889 ± 0.019	[5.841, 5.937]	4.468	4.468 ± 0.015	[4.430, 4.506]
Mixed/UniVer	5.966	5.966 ± 0.018	[5.920, 6.011]	4.521	4.521 ± 0.014	[4.487, 4.555]
DFlash2 with Muse Glimmer 30B						
Top1	3.995	3.995 ± 0.012	[3.966, 4.024]	3.046	3.047 ± 0.007	[3.030, 3.063]
RS	4.808	4.808 ± 0.004	[4.798, 4.818]	3.592	3.592 ± 0.012	[3.562, 3.622]
BlockV	4.986	4.986 ± 0.017	[4.943, 5.028]	3.723	3.723 ± 0.024	[3.664, 3.783]
First	4.871	4.871 ± 0.016	[4.831, 4.912]	3.620	3.620 ± 0.012	[3.592, 3.649]
WoR/RRS	5.211	5.211 ± 0.011	[5.183, 5.239]	3.864	3.864 ± 0.006	[3.850, 3.878]
WoR/Traversal	5.332	5.332 ± 0.017	[5.289, 5.375]	3.953	3.953 ± 0.015	[3.915, 3.992]
Mixed/UniVer	5.304	5.304 ± 0.026	[5.239, 5.369]	3.928	3.928 ± 0.022	[3.873, 3.984]

For a seed-level metric m, the approximate 95% confidence interval (CI) for its mean is

$$\bar{m} \pm t_{0.975,2} \frac{s_m}{\sqrt{3}}, \qquad t_{0.975,2} \approx 4.303, \tag{4}$$

where s_m is the sample SD across seeds, computed with denominator $3 - 1$. These Student-t intervals assume independent, normally distributed seed-level estimates. The six datasets form a fixed evaluation suite; the intervals characterize run-to-run uncertainty on this suite. CIs are centered on the seed mean, while the all-seed pooled result is computed directly from the combined totals.

B.3 Choice of First-Pass Planner

Tsubame does not constrain how the first pass is built: any planner that fixes the topology before replay can be used. We use node-wise max-heap expansion for DFlash2 and EAGLE-2-style expand-then-prune for Domino and DSpark. Both allocate nodes by cumulative draft path probability and differ only in implementation. Max-heap search maximizes summed draft path probability under a node budget (Ringel & Romano, 2026; Lin & Jang, 2026), and expand-then-prune can recover the same solution given a sufficiently wide frontier and candidate pool. We simply retain EAGLE-2's default frontier width of 10; larger frontiers such as 16 or 32 can also be used, and we did not tune this setting.

Table 11 compares DominoTree with our first pass on Domino. The two planners achieve similar acceptance lengths: DominoTree is slightly ahead at $B = 16$, consistent with our narrower frontier, and the difference largely vanishes at $B = 32$ and $B = 64$. Expand-then-prune batches GRU updates and logit corrections across each frontier, requiring at most one sequential correction stage per expansion level, whereas DominoTree's sequential stages scale with the number of expanded nodes (Lin & Jang, 2026); our first pass therefore achieves higher throughput at larger budgets. Mixed/UniVer, built on our first pass, exceeds both planners at every budget, reflecting that Tsubame's gains come from replay rather than from the choice of planner.

Table 11: Domino with Qwen3-8B: DominoTree, First pass, and Mixed/UniVer under stochastic decoding ($T = 1$) across candidate budgets. Budgets include the root, and results pool three seeds.

		Math				Code				Conversation			
		GSM8K		Math500		HumanEval		MBPP		MT-Bench		Alpaca	
B	Method	τ	SR	τ	SR	τ	SR	τ	SR	τ	SR	τ	SR
16	DominoTree	8.08	6.09×	6.81	5.18×	6.56	4.99×	6.30	4.89×	4.28	3.24×	4.00	3.04×
	First	8.00	6.04×	6.75	5.16×	6.49	4.97×	6.30	4.93×	4.26	3.24×	3.97	3.04×
	Mixed/UniVer	**8.57**	**6.44×**	**7.26**	**5.51×**	**6.87**	**5.19×**	**6.48**	**5.02×**	**4.46**	**3.36×**	**4.10**	**3.11×**
32	DominoTree	9.41	6.10×	7.66	5.02×	7.43	4.87×	7.31	4.89×	4.79	3.13×	4.42	2.89×
	First	9.36	6.18×	7.79	5.22×	7.35	4.93×	7.30	5.00×	4.78	3.20×	4.43	2.97×
	Mixed/UniVer	**9.70**	**6.33×**	**8.21**	**5.44×**	**7.80**	**5.17×**	**7.59**	**5.15×**	**5.01**	**3.32×**	**4.62**	**3.06×**
64	DominoTree	10.12	6.35×	8.46	5.35×	8.10	5.12×	8.06	5.20×	5.27	3.33×	4.90	3.10×
	First	10.32	6.81×	8.46	5.67×	8.14	5.45×	8.05	5.51×	5.21	3.48×	4.84	3.25×
	Mixed/UniVer	**10.62**	**6.93×**	**8.85**	**5.87×**	**8.58**	**5.69×**	**8.29**	**5.62×**	**5.44**	**3.60×**	**5.11**	**3.39×**

B.4 Additional Experiments with DSpark and DFlash2–Qwen3.8

DSpark with Qwen3-8B. Replay with Mixed/UniVer improves acceptance length and throughput over First pass on all six datasets (Table 12), with pooled gains of 4.95% and 3.33%, respectively.

Table 12: DSpark with Qwen3-8B results on all six datasets, pooled over three seeds. Chains retain DSpark's native draft depth of 8; trees use a candidate budget of $B = 16$. TopK denotes deterministic candidate selection in the first pass, and Naive denotes target-sample matching. Bold marks the best value in each column.

			Math				Code				Conversation			
			GSM8K		Math500		HumanEval		MBPP		MT-Bench		Alpaca	
Structure	Sampling	Verifier	τ	SR	τ	SR	τ	SR	τ	SR	τ	SR	τ	SR
Chain	Top1	Naive	5.73	4.48×	5.14	4.04×	5.00	3.92×	4.82	3.83×	3.39	2.66×	3.24	2.49×
	Random	RS	6.35	4.84×	5.92	4.56×	5.59	4.34×	5.28	4.13×	3.81	2.93×	3.54	2.68×
	Random	BlockV	6.39	4.89×	6.00	4.65×	5.64	4.37×	5.42	4.21×	3.90	3.01×	3.63	2.76×
Tree (First)	TopK	Naive	6.66	5.03×	6.12	4.63×	5.92	4.50×	5.72	4.43×	4.17	3.16×	3.99	3.02×
Tree (Replay)	WoR	RRS	6.78	5.02×	6.30	4.72×	6.10	4.59×	5.87	4.48×	4.31	3.23×	4.07	3.03×
	WoR	Traversal	6.83	5.08×	**6.38**	**4.81**×	6.16	4.60×	5.94	**4.55**×	4.41	3.30×	4.15	3.09×
	Mixed	UniVer	**6.85**	**5.09**×	**6.38**	4.74×	**6.21**	**4.66**×	**5.98**	**4.55**×	**4.44**	**3.32**×	**4.22**	**3.13**×

DFlash2 with Qwen3.8-27B. The released DFlash2 implementation for Qwen3.8-27B uses SGLang (Zheng et al., 2024), whereas our Muse experiments use the Python implementation. We therefore compare methods using DFlash2's native SGLang single-request configuration and a matched SGLang AR baseline, reporting request-mean acceptance length and SR. As shown in Table 13, Mixed/UniVer improves the six-dataset request-mean acceptance length by 1.84% and throughput by 1.25% over First pass. On code tasks, sampled chains with RS yield shorter acceptance lengths than Top1 chains (e.g., 4.26 vs. 4.40 on HumanEval), indicating that sampling a lone candidate is not beneficial for this model–task pair. We therefore fill singleton child slots with the top-1 token on code tasks, while nodes with multiple children retain mixed sampling (top-$(m-1)$ candidates plus one sampled candidate). This choice follows the chain-level comparison between RS and Top1, not tree-level results.

Table 13: DFlash2 with Qwen3.8-27B (non-thinking) results on all six datasets, pooled over three seeds. Chains retain Qwen3.8-DFlash2's native draft depth of 8; trees use a candidate budget of $B = 16$. Results use the native SGLang single-request configuration and report request-mean acceptance length and SR relative to a matched SGLang AR baseline. TopK denotes deterministic candidate selection in the first pass, and Naive denotes target-sample matching. Bold marks the best value in each column.

			Math				Code				Conversation			
			GSM8K		Math500		HumanEval		MBPP		MT-Bench		Alpaca	
Structure	Sampling	Verifier	τ_{req}	SR	τ_{req}	SR	τ_{req}	SR	τ_{req}	SR	τ_{req}	SR	τ_{req}	SR
Chain	Top1	Naive	4.26	3.15×	5.57	4.31×	4.40	3.21×	3.46	2.55×	3.40	2.49×	2.85	2.20×
	Random	RS	4.39	3.22×	5.79	4.50×	4.26	2.96×	3.40	2.41×	3.70	2.71×	3.08	2.35×
	Random	BlockV	4.44	3.26×	5.88	4.57×	4.37	3.04×	3.43	2.44×	3.73	2.74×	3.14	2.40×
Tree (First)	TopK	Naive	4.81	3.38×	6.38	4.71×	4.84	3.35×	3.99	2.81×	4.11	2.88×	3.45	2.53×
Tree (Replay)	WoR	RRS	4.80	3.38×	6.35	4.63×	4.80	3.23×	4.03	2.76×	4.15	2.91×	3.51	2.56×
	WoR	Traversal	4.89	**3.46**×	6.39	4.67×	4.80	3.24×	3.94	2.69×	4.17	2.93×	3.56	2.57×
	Mixed	UniVer	**4.91**	3.42×	**6.47**	**4.74**×	**4.90**	**3.38**×	**4.04**	**2.82**×	**4.23**	**2.96**×	**3.56**	**2.59**×

B.5 TEMPERATURE ABLATION

Table 14: Domino with Qwen3-8B at different target temperatures at $B = 16$, pooled over six datasets. All draft proposals use sparse64 candidate pools. Stochastic results combine three seeds. Greedy ($T = 0$) results use seed 0 and are included only for Top1 chain and First pass; stochastic sampling and replay entries are omitted (–). Bold marks the best value in each stochastic-decoding column. The underlined verifier denotes the default configuration adopted by Domino.

			$T = 1.0$		$T = 0.6$		$T = 0$	
Structure	Sampling	Verifier	τ	TPS	τ	TPS	τ	TPS
Chain	Top1	Naive	4.54	126.0	5.27	145.9	5.82	161.2
	Random	RS	5.07	139.6	5.54	152.3	–	–
	Random	BlockV	5.29	145.4	5.69	156.2	–	–
Tree (First)	TopK	Naive	5.68	155.1	6.35	173.4	6.85	187.2
Tree (Replay)	WoR	RRS	5.72	155.3	6.32	171.2	–	–
	WoR	Traversal	5.89	159.8	6.39	172.9	–	–
	Mixed	UniVer	**5.97**	**161.6**	**6.48**	**175.2**	–	–

Table 15: DFlash2 with Muse Glimmer 30B at different target temperatures at $B = 16$, pooled over six datasets. All draft proposals use DFlash2's native 16-token candidate pools. Stochastic results combine three seeds. Greedy ($T = 0$) results use seed 0 and are included only for Top1 chain and First pass; stochastic sampling and replay entries are omitted (–). Bold marks the best value in each stochastic-decoding column. The underlined verifier denotes the default configuration adopted by DFlash2.

			$T = 1.0$		$T = 0.6$		$T = 0$	
Structure	Sampling	Verifier	τ	TPS	τ	TPS	τ	TPS
Chain	Top1	Naive	3.99	69.8	4.52	78.9	5.00	87.4
	Random	RS	4.81	82.3	4.99	85.7	–	–
	Random	BlockV	4.99	85.3	5.11	87.5	–	–
Tree (First)	TopK	Naive	4.87	83.0	5.45	92.4	5.85	99.6
Tree (Replay)	WoR	RRS	5.21	88.6	5.54	94.0	–	–
	WoR	Traversal	**5.33**	**90.6**	5.61	95.3	–	–
	Mixed	UniVer	5.30	90.0	**5.63**	**95.6**	–	–

Temperature ablation. Domino and DFlash2–Muse exhibit similar temperature trends, although the magnitudes differ (Tables 14 and 15). At $T = 1$, the probability distributions are more diffuse, giving sampling-based chains and trees a larger advantage over deterministic candidate selection. As temperature decreases and probability mass concentrates, this advantage diminishes. The greedy $T = 0$ entries use seed 0; in this limit the distributions collapse to point masses and verification reduces to token matching.

B.6 Latency Breakdown

Table 16 reports the latency breakdown using the first eight samples from each of GSM8K, HumanEval, and MT-Bench at $B = 16$ and $T = 1$. Domino runs on an RTX 3090; DFlash2–Muse runs on an RTX PRO 6000.

Table 16: Latency in ms per steady-state decoding round, pooled over the three datasets. Tree combines construction and replay.

Method	Target	Draft	Others	Verifier	Tree	Total
Domino with Qwen3-8B (RTX 3090)						
First pass	27.951	5.875	0.843	0.131	0.993	35.794
WoR/RRS	27.848	5.868	0.817	0.128	1.274	35.935
WoR/Traversal	27.889	5.897	0.834	0.164	1.274	36.058
Mixed/UniVer	27.926	5.865	0.832	0.164	1.292	36.079
DFlash2 with Muse Glimmer 30B (RTX PRO 6000)						
First pass	49.246	7.732	1.104	0.159	0.506	58.747
WoR/RRS	49.252	7.734	1.083	0.149	0.572	58.789
WoR/Traversal	49.237	7.743	1.084	0.170	0.572	58.807
Mixed/UniVer	49.204	7.741	1.083	0.173	0.590	58.791

Domino incurs a higher second-pass tree construction cost because its GRU-based correction must be rerun under the replayed prefixes, whereas DFlash2 only requires transition-score lookups and makes replay particularly inexpensive. Verification can add a small amount of latency as well. Together, the additional tree construction and verification costs are on the order of 1% of Domino's total round latency, and are even smaller for DFlash2.

B.7 Sparse Candidate Pools for Domino

As described in Section 3.1, Domino restricts its draft vocabulary to the 64 highest-scoring backbone tokens at each position (sparse64) and applies GRU corrections and normalization within this pool; verification and the output distribution remain full-vocabulary. We apply sparse64 to all Domino methods, including chains, so that chain and tree proposals share the same support. Table 17 shows that this does not weaken the chain baselines: sparse64 changes pooled acceptance length by less than 0.7% for all three chains while increasing speedup ratio (SR) by 1.9%–4.7%.

Table 17: Domino chain baselines with sparse64 and full-vocabulary draft proposals at $T = 1$ and $B = 16$, pooled over six datasets and three seeds. SR is relative to the matched Qwen3-8B AR baseline, and changes are relative to full-vocabulary drafting.

	Acceptance length			SR ($\times$)		
Method	Sparse64	Full vocab.	Change	Sparse64	Full vocab.	Change
Top1	4.539	4.570	-0.67%	3.524	3.458	$+1.90\%$
RS	5.065	5.060	$+0.11\%$	3.904	3.730	$+4.66\%$
BlockV	5.291	5.308	-0.32%	4.066	3.902	$+4.20\%$

C LIMITATIONS

First, Tsubame primarily improves stochastic decoding ($T > 0$). At $T = 0$, target decoding involves no sampling and verification reduces to deterministic token matching, removing the sampling-based acceptance gains that motivate replay. Second, replay relies on inexpensive candidate regeneration through lightweight conditional corrections. The cost of these operations varies across drafters, so the resulting throughput gains depend on how efficiently each architecture supports the second pass.

D LICENSES

Table 18: Licenses of code, model weights, and evaluation datasets. Resource names link to their upstream releases. "Not specified" indicates that the linked release does not state an explicit license.

Category	Resource	License
Code	Domino	Not specified
	DSpark (DeepSpec)	MIT
	DFlash2	MIT
Target weights	Qwen3-8B	Apache-2.0
	Muse Glimmer 30B	Apache-2.0
	Qwen3.8-27B	Apache-2.0
Draft weights	Domino for Qwen3-8B	Not specified
	DSpark for Qwen3-8B	Not specified
	DFlash2 for Muse Glimmer 30B	Apache-2.0
	DFlash2 for Qwen3.8-27B	Apache-2.0
Datasets	GSM8K	MIT
	Math500 (PRM800K split)	MIT
	HumanEval	MIT
	MBPP	CC BY 4.0
	MT-Bench questions (FastChat)	Apache-2.0
	Alpaca	CC BY-NC 4.0