D^{2}SD: ACCELERATING SPECULATIVE DECODING WITH DUAL DIFFUSION DRAFT MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates autoregressive large language model inference by drafting multiple tokens and verifying them in a single target-model forward pass. Recent diffusion-based drafters generate an entire block of tokens in parallel but usually commit to a single draft sequence per verification: once the first mismatch occurs, all subsequent draft tokens are discarded, resulting in a limited acceptance rate. Naively batching more draft candidate sequences can yield limited acceptance gains, as redundant or poorly placed branches increase the cost of drafting and verification without proportionally increasing the number of accepted tokens. We propose D^{2}SD, a dual diffusion draft speculative decoding framework that organizes candidates into a confidence-guided prefix tree, where the first diffusion drafter generates a block along with per-position confidence scores that are used to identify the most likely rejection boundary and select the top-K prefix ranges for recovery; the second variable-prefix diffusion drafter re-anchors at each selected prefix and proposes alternative continuations in one batched pass; the resulting shared-prefix candidates are jointly verified via cascade attention. Empirically, D^{2}SD shows clear improvements over both the underlying diffusion approach and strong autoregressive speculative decoding baselines.

1 INTRODUCTION

Autoregressive large language models (LLMs) achieve strong performance (Liu et al., 2024; Yang et al., 2025; Zeng et al., 2026), but the decoding computation remains inherently serial and memory-bandwidth bounded: each new token requires one forward pass through the original model. Speculative decoding alleviates this bottleneck by exploiting idle GPU compute to parallelize generation with a cheaper drafter, to propose multiple tokens in a single verification step over the original model (a.k.a. target model) (Leviathan et al., 2023; Chen et al., 2023).

One fundamental challenge of speculative decoding lies in the sequential nature of its verification protocol: due to the *longest-correct-prefix rule* (i.e., the speedup is determined by the longest prefix of the draft token sequence(s) accepted via verification), a single mismatch at an early position i immediately invalidates all subsequent draft tokens in the sequence, regardless of their individual correctness. Note that this inherited issue creates a non-uniform importance across drafting positions—errors near the "anchor" (i.e., the last verified token) are catastrophic, while errors near the end of a block could be negligible. Consequently, simply increasing the drafting budget (i.e., generating more tokens) does not guarantee end-to-end speedup unless the additional compute is strategically allocated to minimize early rejections.

To address this, prefix tree-based speculative decoding (e.g., SpecInfer (Miao et al., 2024), Sequoia (Chen et al., 2024), and EAGLE-3 (Li et al., 2025b)) organizes candidate sequences into branching structures to explore multiple continuations. While effective at increasing acceptance length, these methods are fundamentally limited by their autoregressive drafting phase. This sequential generation imposes a heavy "drafting tax" that scales poorly with the candidate budget: as the token budget increases from 4 to 16, the drafting overhead can balloon from 7ms to 24ms (Chen et al., 2026).

Conversely, parallel drafting methods like DFlash (Chen et al., 2026) leverage a diffusion draft model to generate an entire block of tokens in a single forward pass, drastically reducing drafting

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.

latency. However, existing diffusion-style drafters typically apply a uniform computational budget across all positions within a block, although later positions face increasing predictive difficulty. Our empirical study reveals that this computational mismatch leads to a "scaling wall" (Appendix C.1, Table 6): simply increasing the block size γ yields diminishing or even negative returns once $\gamma \geq 24$, as an early error in a long, linear block renders the remaining extended budget useless. This bottleneck motivates our investigation into *how to accelerate speculative decoding using diffusion-based draft model(s) with sophisticated construction and organization of draft sequences*.

Enabling parallel drafting with tree-structured budget allocation is non-trivial. The most intuitive baseline—generating K independent candidates by sampling from the same diffusion drafter forward pass—offers limited acceptance gains. These candidates exhibit high error homogeneity: because they share the same underlying probability distribution, they tend to collapse onto the same high-confidence paths and collectively fail at the same difficult decision boundaries. As shown in our ablation (Table 7, full results in Section 4.3), adding $K{=}4$ naive branches to DFlash on GSM8K improves the acceptance length by only 0.37 tokens (TPF $6.96{\rightarrow}7.33$), whereas D^{2}SD lifts it by 2.25 tokens (to 9.21) under the same branching budget. This underscores that effective branching requires more than just increasing the sample count; it requires statistically informed recovery at likely rejection points using a model with a distinct inductive bias.

Towards this end, we propose D^{2}SD, a dual diffusion drafting paradigm to accelerate speculative decoding, where our method differs by explicitly constructing a prefix-tree-like candidate set from the first diffusion drafter's confidence scores, so that re-drafting by the second specially trained drafter is performed only at likely rejection boundaries rather than through uniform, heuristic, or computationally expensive branching. Concretely, we make the following contributions:

Contribution 1. We design a dual-draft paradigm: First, a DFlash-style diffusion drafter generates an entire block in one parallel pass and produces per-position confidence scores. These confidence scores are then used to estimate the most likely rejection locations and to select the top-K prefix ranges with the most promising recovery. Second, we introduce a variable-prefix diffusion drafter that re-anchors at those selected prefix positions and proposes alternative suffixes in parallel, thereby converting a single diffusion draft into a structured set of shared-prefix candidates.

Contribution 2. We implement this dual-draft paradigm with specialized training recipes and optimizations: To support this second stage, we design a dedicated training recipe that exposes the model to variable-length visible prefixes and applies an exponentially decaying training loss that emphasizes errors near the anchor. To enable efficient speculative decoding, we verify the resulting branches using cascade attention, allowing the method to reuse shared-prefix computation rather than replicating long KV prefixes across candidates.

Contribution 3. We conduct a concrete empirical study to demonstrate clear improvements over strong speculative decoding baselines. For example, on Qwen3-8B model with a default configuration of ($\gamma{=}16$) and ($K{=}4$), D^{2}SD increases the average greedy-decoding speedup from ($4.16\times$) to ($4.98\times$) relative to standard autoregressive decoding, while raising the average acceptance length from 5.31 to 7.05 when compared with DFlash.

2 PRELIMINARIES

Speculative decoding speedup. Speculative decoding accelerates autoregressive inference of a target model $\mathcal{M}_t$ by introducing a lightweight draft model $\mathcal{M}_d$. In each decoding cycle, $\mathcal{M}_d$ proposes $\gamma{-}1$ candidate tokens in a block whose position 0 is the anchor, which $\mathcal{M}_t$ verifies in a single parallel forward pass and accepts up to the longest correct prefix. Following Sadhukhan et al. (2025), the average per-token latency of speculative decoding can be formulated as:

$$L \;=\; \frac{T_{\text{draft}} + T_{\text{verify}}}{\alpha}, \tag{1}$$

where T_{draft} is the time spent producing the draft block, T_{verify} is the cost of one verification pass, and $\alpha \in [1, \gamma]$ is the expected number of tokens committed per cycle (the mean acceptance length, including the bonus token returned by $\mathcal{M}_t$ at the rejection boundary). Letting L_{target} denote the per-token latency of standard autoregressive decoding with $\mathcal{M}_t$, the resulting wall-clock speedup is

$$\eta \;=\; \frac{L_{\text{target}}}{L} \;=\; \frac{\alpha \cdot L_{\text{target}}}{T_{\text{draft}} + T_{\text{verify}}}. \tag{2}$$

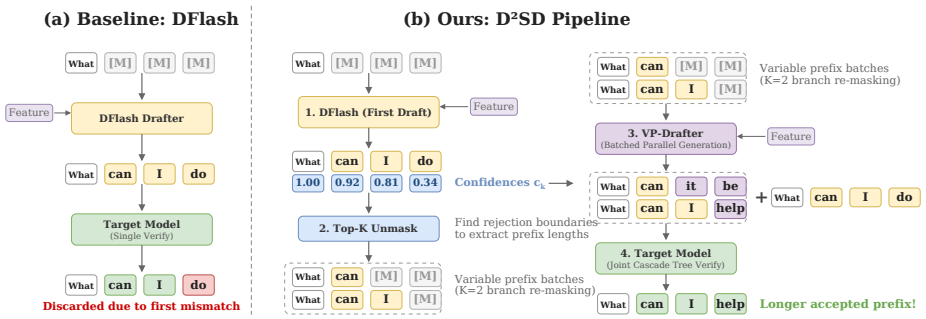


Figure 1: **(a) Baseline DFlash** on a $\gamma = 4$ block: the drafter consumes the bonus-token anchor and $\gamma - 1$ mask positions, and predicts all mask positions in one pass conditioned on target features. The target accepts the correct prefix and discards the remainder. **(b) Our D^{2}SD pipeline**, with $K = 2$: DFlash produces tokens and confidences; Top-K Unmask selects likely rejection boundaries; the VP-Drafter regenerates their suffixes in one batched pass; the target jointly verifies all $K + 1$ candidates using cascade attention.

This expression makes the trade-off explicit: speedup grows either by raising the acceptance length α or by reducing the per-cycle drafting/verification overhead $T_{\text{draft}}+T_{\text{verify}}$. Block-diffusion drafters such as DFlash (Chen et al., 2026) drive T_{draft} down by emitting a γ-token block in a single forward pass, but their reliance on a single draft sequence caps α well below γ once an early mismatch occurs. D^{2}SD targets exactly this gap: it raises α by re-drafting at the predicted rejection boundary while amortizing the additional work through a single batched second-draft pass and a shared-prefix joint verification.

DFlash. DFlash (Chen et al., 2026) is a block-diffusion speculative decoding framework that fills a γ-position block in a single parallel forward pass; panel (a) of Figure 1 illustrates one decoding cycle as a drafter $\rightarrow$ target flow. *Drafting*: The drafter consumes a length-γ input whose position 0 holds the anchor—the bonus token returned by the previous verification step—and whose positions $1, \ldots, \gamma-1$ are all [MASK]. Conditioned on the target hidden features of the most recently accepted tokens—multi-layer target representations concatenated and projected by a learned FC layer, then injected into the Key and Value projections of every drafter layer—the drafter decodes all $\gamma-1$ mask positions jointly in one parallel pass, with mask tokens attending bidirectionally to the target's representations. We highlight that the target model then verifies the drafted block by prefix matching, accepting tokens up to (but not including) the first mismatch and discarding the rest; the same forward pass also returns the next anchor (the target's prediction at the rejection boundary) and refreshed target hidden features that seed the next cycle.

Limitation. DFlash acceptance length is fundamentally bounded by the *longest correct prefix*: once a mismatch occurs at some position, all subsequent draft tokens are discarded regardless of their quality. This observation motivates our dual-draft design, which re-drafts from the rejection boundary to recover these otherwise-lost positions.

3 D^{2}SD

We present D^{2}SD (Dual-Draft Diffusion Speculative Decoding), designed to address the two failure modes identified in the introduction: the scaling wall of single-block diffusion drafting and the error homogeneity of resampling branches from the same DFlash forward pass. Rather than extend the draft block, we re-anchor it at positions where the first draft is most likely to fail; rather than reuse DFlash for the second branch, we introduce a separately trained VP-Drafter (Variable-Prefix Drafter), giving the cascade a different inductive bias.

3.1 Estimating the Rejection Boundary from Drafter Confidence

D^{2}SD replaces the static allocation in DFlash with a dynamic, difficulty-aware one: we re-anchor a second drafter at the predicted rejection boundary, directing an additional forward pass and a distinct inductive prior to positions where the first draft is most likely to fail. Re-drafting too early wastes capacity on positions that would already have been accepted, while re-drafting too late leaves the actual rejection inside the kept prefix. Since the rejection boundary is unknown until verification, the second drafter must predict where the first draft will fail.

We use the first drafter's own per-position probability mass as a target-free signal. At each cycle DFlash produces a categorical $p_k(\cdot)$ at every block position $k = 1, \ldots, \gamma{-}1$ and we greedy-sample $\hat{t}_k = \arg\max_v p_k(v)$; we define the *confidence* of position k as

$$c_k \;=\; p_k(\hat{t}_k) \;=\; \max_v p_k(v). \tag{3}$$

Figure 2a shows the original GSM8K confidence–acceptance diagnostic, where the empirical accept rate broadly tracks confidence. Similar signals have been studied for autoregressive drafters (Du et al., 2024; Li et al., 2024b). Using c_k as a surrogate for $\Pr(\hat{t}_k \text{ accepted} \mid \hat{t}_{<k} \text{ accepted})$, we form the following approximation to the first-rejection distribution:

$$r(i) \;=\; \underbrace{\prod_{k=1}^{i} c_k}_{\text{first } i \text{ accepted}} \;\cdot\; \underbrace{(1 - c_{i+1})}_{\text{position } i{+}1 \text{ rejected}}, \quad i = 0, 1, \ldots, \gamma{-}2. \tag{4}$$

This confidence-based approximation ranks rejection boundaries using the first drafter's outputs. The remaining mass $\prod_{k=1}^{\gamma-1} c_k$ corresponds to full-block survival. Boundary selection uses the drafter outputs directly, and Section 4.4 evaluates its predictive accuracy.

3.2 Top-K Shared-Prefix Branch Selection

Effective branching must answer two separate questions: where each branch should diverge from the first draft, and which model produces its continuation. We answer the first question using $r(i)$; the second—a VP-Drafter trained with a different inductive bias—is addressed in Section 3.4.

Committing to the single $\arg\max_i r(i)$ can miss recovery opportunities, because $r(i)$ is typically diffuse—several prefix lengths receive comparable mass. We therefore retain the top-K prefix lengths,

$$\mathcal{S} \;=\; \underset{i \in \{0,\ldots,\gamma-2\}}{\text{Top-K}} \; r(i), \tag{5}$$

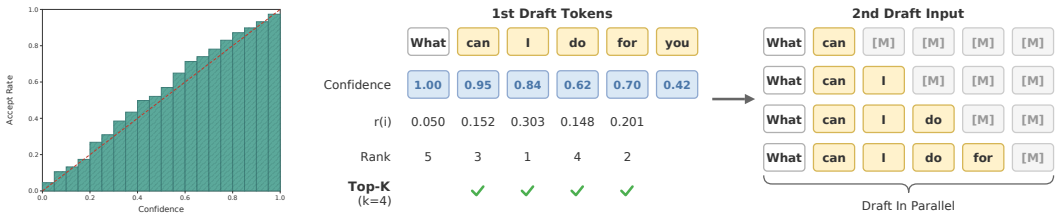
and launch one second-draft branch from each. Branch $i \in \mathcal{S}$ retains the anchor and the first i DFlash tokens as its visible prefix and re-drafts the remaining $\gamma{-}1{-}i$ positions with the VP-Drafter (Figure 2b). By construction, every branch shares its first $i{+}1$ tokens with the corresponding DFlash prefix, so the resulting candidate set has exactly the shared-prefix structure that cascade-style joint verification (Section 3.3) can exploit—inheriting the verification-amortization benefit of tree-based speculative decoding (Miao et al., 2024; Chen et al., 2024; Li et al., 2024b; Cai et al., 2024) without paying their autoregressive drafting tax.

3.3 End-to-End Inference Pipeline

A full D^{2}SD decoding cycle combines the DFlash first draft, the VP-Drafter second draft, and joint verification. Each cycle consumes the verified KV caches of the target and the two drafters, along with the bonus token from the previous cycle, and emits a new batch of accepted tokens. Panel (b) of Figure 1 illustrates one such cycle on a $\gamma = 4$ block; the four stages are described below.

(i) First draft (DFlash). As in standard DFlash, the drafter builds a length-γ masked block: position 0 holds the bonus token (the anchor) and positions $1, \ldots, \gamma{-}1$ are [MASK]. Conditioned on the target hidden features from the previous cycle, DFlash predicts all $\gamma - 1$ mask positions in one pass, yielding tokens $\hat{t}_1, \ldots, \hat{t}_{\gamma-1}$ with confidences $c_1, \ldots, c_{\gamma-1}$.

(ii) Top-K unmask. From the confidences we compute $r(i)$ (Eq. 4) for each $i \in \{0, \ldots, \gamma{-}2\}$ and pick the top-K prefix lengths $\mathcal{S}$ (Eq. 5); each $i \in \mathcal{S}$ becomes a re-anchoring position.



(a) Confidence vs. accept rate.

(b) Top-K unmask procedure.

Figure 2: **(a)** DFlash drafter confidence versus empirical accept rate on GSM8K; each bar averages draft tokens in a confidence bin, and the dashed line is $y = x$. **(b)** Top-K unmask with $K = 4$ on a $\gamma = 6$ block: score each prefix length with $r(i)$ and select the four highest. Each selected prefix retains its tokens and re-masks the remainder; all branches are drafted in parallel.

(iii) Second draft (VP-Drafter, batched). For every $i \in \mathcal{S}$, the second-draft input retains the anchor and the first i DFlash tokens as the visible prefix and re-masks positions $i{+}1, \ldots, \gamma{-}1$. The K branches are stacked into a single batch and processed by the VP-Drafter in one parallel pass, reusing the same target hidden features as DFlash. Each output branch agrees with DFlash on its first $i{+}1$ tokens and proposes alternative continuations for the remainder.

(iv) Joint verification and acceptance. Together with the original DFlash branch, the target model jointly verifies the $K{+}1$ candidates in a single forward pass. For greedy decoding we commit the longest target-consistent prefix; the target's prediction at the rejection position becomes the next anchor (the bonus token). For sampling, we make one target draw at each visited prefix and retain the branches agreeing with it. This preserves the target distribution under exact target conditionals (Appendix F). Cascade attention shares the verified context KV while retaining branch-local candidate blocks.

3.4 Training the VP-Drafter

The VP-Drafter shares its architecture with DFlash—a lightweight Qwen-based block-diffusion model with KV-injected target features—but uses a different masking schedule designed to support *any* prefix length within the block.

At inference, the unmask procedure of Section 3.2 draws K different prefix lengths from $\{0, \ldots, \gamma{-}2\}$, and the second drafter must produce sensible continuations from each of them. DFlash, by contrast, is trained with a single fixed anchor at position 0; reusing it for the second stage would force extrapolation to unseen prefix lengths, changing the input distribution relative to its fixed-anchor training.

For each training example we sample a prefix length l from a truncated geometric prior

$$\Pr(l = j) \;\propto\; \beta^j, \quad j = 0, 1, \ldots, \gamma{-}2, \tag{6}$$

with $\beta \in (0, 1)$ biasing toward shorter prefixes. The first $l{+}1$ positions are filled with ground-truth tokens and positions $l{+}1, \ldots, \gamma{-}1$ are masked. The drafter then predicts the masked positions in parallel, conditioned on target hidden features extracted from the same teacher pass used by DFlash. Because verification accepts tokens sequentially from the anchor outward, errors on positions closer to the anchor are strictly more costly than errors on later positions. We therefore train with an exponentially decayed cross-entropy that weights each masked position by its distance from the anchor,

$$\mathcal{L}_{\text{VP}} \;=\; -\frac{\sum_{k=l+1}^{\gamma-1} w_k \, \log p_k(t_k^\star)}{\sum_{k=l+1}^{\gamma-1} w_k}, \qquad w_k \;=\; \exp\!\left(-\frac{k-l-1}{\tau}\right), \tag{7}$$

with the expectation taken over data and the prior in Eq. (6). We fix $\beta = 0.9$ and $\tau = 16$ across benchmarks. The two drafters are trained independently from scratch. Fixed K specifies the compute budget, while the selected prefix locations change each cycle.

4 EXPERIMENTS

4.1 SETUP

Models. We use Qwen3-8B and GPT-OSS-20B as the target model throughout. The main benchmarks use NVIDIA H200 GPUs with one request at a time. Supplementary boundary and concurrency experiments use B300 GPUs; we report these separately.

Benchmarks. We evaluate on eight tasks spanning three categories: **Math**—GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), and AIME25 (Mathematical Association of America, 2025); **Code**—HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and LiveCodeBench (Jain et al., 2025); **Chat**—MT-Bench (Zheng et al., 2023) and Alpaca (Taori et al., 2023).

Training data. All the EAGLE-3, DFlash and the VP-Drafter are trained on the full PerfectBlend[1] dataset (Xu et al., 2024), which aggregates diverse instruction–response pairs across mathematics, code, chat, and instruction following domains, unless otherwise noted. To ensure the draft models are aligned with the target distribution, all training responses are generated by the target model itself.

Metrics. We measure the *mean acceptance length* α, defined as the average number of tokens accepted per decoding cycle, and the *wall-clock speedup*, defined as the ratio of the autoregressive baseline's per-token decode latency to that of the speculative method.

Implementation. All models are served in BF16 precision with FlashAttention-2 (Dao, 2024). Cascade attention is implemented with flashinfer (Ye et al., 2025) kernels and pre-warmed CUDA graphs for both the VP-Drafter and the target verification pass. The default configuration uses second-draft branch sizes γ=16 and K=4. We use HuggingFace Transformers as the model backbone and train the models on the SpecForge[2].

Training budget. In the controlled Qwen3 comparison, each drafter receives 266,000 optimizer updates with global batch 32. DFlash and VP-Drafter are trained independently from scratch, so D^{2}SD requires two training runs. Each architecture uses its specified sequence length and training objective. Appendix A gives full settings and compute accounting. The GPT-OSS comparison instead uses a public first-draft checkpoint.

4.2 END-TO-END RESULTS

Tables 1 and 2 report wall-clock speedup and mean acceptance length for DFlash, EAGLE-3 (Li et al., 2025b), and D^{2}SD ($K = 4$, $\gamma = 16$) on Qwen3-8B and GPT-OSS-20B under greedy ($T = 0$) and multinomial ($T = 1$) decoding. Against the autoregressive EAGLE-3, D^{2}SD wins on wall-clock speedup on every task even when EAGLE-3 occasionally reaches a slightly longer α, showing the latency advantage of parallel block drafting. On Qwen3-8B, average greedy speedup rises from $4.16\times$ to $4.98\times$ over autoregressive decoding and acceptance from 5.31 to 7.05; at $T = 1$, speedup rises from $3.43\times$ to $4.01\times$. On GPT-OSS-20B, average greedy speedup rises from $3.53\times$ to $6.15\times$. At $T = 1$, average speedup rises from $1.71\times$ to $1.84\times$. Per-task speedup reflects both acceptance length and the cost of drafting and verification.

4.3 ABLATION STUDY OF D^{2}SD

To justify the design of D^{2}SD, we conduct the following ablation studies on Qwen3-8B with greedy target decoding.

Ablation 1. *Why is augmenting DFlash with K resampled branches insufficient?* A natural baseline for the VP-Drafter is to skip the second model and let DFlash itself produce extra candidates: we keep the original argmax draft and append K=4 branches drawn as per-position T=1 multinomial samples from the same drafter forward, verifying all K+1 jointly via cascade attention. This matches D^{2}SD's branching budget, so any difference reflects the source of branch diversity rather than verifier capacity. The resampled branches recover only a small slice of the headroom—average α rises from 5.75 to 6.08 and speedup from $4.48\times$ to $5.06\times$, well short of the $\alpha = 7.62$ that D^{2}SD

[1] https://huggingface.co/datasets/mlabonne/open-perfectblend

[2] https://github.com/sgl-project/SpecForge

Table 1: Qwen3-8B on H200. Each cell is speedup over autoregressive decoding ($\times$) / mean committed tokens per cycle (α). $\gamma = 16$, $K = 4$ for D^{2}SD. Averages are arithmetic means over the eight tasks.

	Greedy ($T = 0$)			Sampling ($T = 1$)		
Task	DFlash	EAGLE-3	D^{2}SD	DFlash	EAGLE-3	D^{2}SD
GSM8K	5.51/6.96	4.14/6.36	6.54/9.21	4.62/5.84	3.72/5.93	5.32/7.38
MATH	5.11/6.55	4.11/6.44	6.03/8.56	4.25/5.38	3.81/6.07	4.90/6.81
AIME25	4.32/5.49	4.03/6.33	5.10/7.26	3.55/4.53	3.57/5.73	4.11/5.75
HumanEval	4.39/5.50	3.85/5.92	5.37/7.47	3.49/4.35	3.47/5.59	4.05/5.50
MBPP	4.07/5.09	3.92/6.03	5.03/6.96	3.25/4.03	3.50/5.57	4.13/4.88
LiveCodeBench	3.88/4.86	3.70/5.93	4.62/6.44	3.06/3.85	3.10/5.01	3.53/4.90
MT-Bench	3.23/4.41	3.35/5.22	3.80/5.75	2.69/3.52	2.79/4.40	3.09/4.36
Alpaca	2.79/3.59	3.30/5.09	3.38/4.74	2.50/3.17	2.73/4.33	2.97/4.02
Average	4.16/5.31	3.80/5.91	4.98/7.05	3.43/4.33	3.34/5.33	4.01/5.45

Table 2: GPT-OSS-20B on H200. Each cell is speedup over autoregressive decoding ($\times$) / mean committed tokens per cycle (α). $\gamma = 16$, $K = 4$ for D^{2}SD. Averages are arithmetic means over the eight tasks.

	Greedy ($T = 0$)			Sampling ($T = 1$)		
Task	DFlash	EAGLE-3	D^{2}SD	DFlash	EAGLE-3	D^{2}SD
GSM8K	2.83/3.58	2.32/2.81	6.20/8.02	1.65/2.00	1.17/1.45	1.94/2.49
MATH	2.48/3.01	2.33/2.84	5.43/7.03	1.90/2.29	1.18/1.46	2.27/2.91
AIME25	1.83/2.14	2.26/2.74	5.18/6.51	1.89/2.21	1.16/1.44	2.07/2.54
HumanEval	3.50/4.09	2.44/2.99	6.33/7.86	1.71/1.97	1.17/1.44	1.93/2.37
MBPP	5.33/6.21	2.44/2.96	8.87/11.94	1.65/1.93	1.18/1.45	1.69/2.23
LiveCodeBench	4.72/5.54	2.42/2.93	6.64/9.02	1.68/1.97	1.19/1.45	1.62/2.17
MT-Bench	2.65/3.01	2.23/2.70	2.67/3.44	1.60/1.80	1.15/1.44	1.59/2.04
Alpaca	4.88/5.49	2.36/2.87	7.87/10.37	1.58/1.78	1.14/1.41	1.58/2.05
Average	3.53/4.13	2.35/2.85	6.15/8.02	1.71/1.99	1.17/1.44	1.84/2.35

attains. The reason is structural: all $K+1$ branches share one drafter forward and therefore the same per-position categorical p_k. At confident positions many samples repeat the kept argmax, while uncertain positions remain governed by the same per-position distribution. Resampling the same drafter thus leaves headroom for a model with a different inductive bias—the VP-Drafter, trained for variable-length prefixes (Section 3.4).

Ablation 2. *Why not directly reuse DFlash as the second drafter?* The complementary baseline keeps D^{2}SD's full pipeline—top-K confidence unmask, variable-prefix re-anchoring, and joint cascade verification—but reuses the original DFlash model as the second drafter, so the only change from D^{2}SD is the inductive bias of the second draft. Because DFlash is trained with a fixed anchor at position 0, applying it to the variable-length prefixes the cascade selects forces extrapolation to settings it has never seen.

The cascade structure clearly helps: average α rises to 6.53 ($+14\%$ over single-branch DFlash, against only $+6\%$ for the naive resampling baseline), confirming that re-anchoring at the predicted rejection boundary places branches more effectively than uniform sampling. Wall-clock speedup, however, rises only modestly to $4.69\times$ ($+5\%$), since the second drafter is now a full forward pass rather than a logits view of the first. Full D^{2}SD still leads by 1.09 tokens of α ($+17\%$) and $0.66\times$ of speedup ($+14\%$)—supporting the variable-prefix training of Eq. (7).

Ablation 3. *Why not use more cascaded levels?* A natural question is whether cascading the same idea once more buys further headroom. We test this by stacking a third VP-Drafter level on top of D^{2}SD: for every $i \in \mathcal{S}$, we re-apply the rejection-boundary estimator (Eq. 4) to that branch's second-draft tail, take the single top-1 location j_i, and dispatch a third VP-Drafter forward anchored there. This yields up to $2K+1$ shared-prefix candidates that the target jointly verifies in a single

Table 3: Per-task Qwen3-8B controls on H200. Cells show speedup ($\times$) / α, with greedy target decoding. Samples appends four independent draws from the first pass; DF→DF uses DFlash for both passes.

Task	DFlash	+4 samples	DF→DF	D^2SD	+third level
GSM8K	5.51/6.96	6.18/7.33	5.76/7.95	6.54/9.21	6.25/9.89
MATH	5.11/6.55	5.67/6.82	5.28/7.35	6.03/8.56	5.79/9.27
MBPP	4.07/5.09	4.66/5.47	4.36/5.90	5.03/6.96	4.85/7.58
MT-Bench	3.23/4.41	3.74/4.70	3.35/4.91	3.80/5.75	3.63/6.20
Average	4.48/5.75	5.06/6.08	4.69/6.53	5.35/7.62	5.13/8.24

Table 4: MATH-500 throughput in generated tokens/s/GPU on B300. Parentheses show ratios to DFlash at the same concurrency B. Each cell uses 128 requests, at most 256 new tokens, $\gamma = 16$, and $T = 0$.

Method	$B = 1$	$B = 2$	$B = 4$	$B = 8$	$B = 16$
DFlash	178.0 (1.00)	134.8 (1.00)	81.1 (1.00)	66.4 (1.00)	66.2 (1.00)
D^2SD, $K = 2$	213.8 (1.20)	152.9 (1.13)	89.0 (1.10)	72.6 (1.09)	73.5 (1.11)
D^2SD, $K = 4$	228.0 (1.28)	169.6 (1.26)	94.2 (1.16)	75.3 (1.13)	75.5 (1.14)
D^2SD, $K = 8$	229.1 (1.29)	166.2 (1.23)	94.8 (1.17)	76.5 (1.15)	77.8 (1.17)

pass; reusing the VP-Drafter is natural since its variable-prefix training (Section 3.4) already covers any prefix length in $\{1, \ldots, \gamma-2\}$.

The third level buys acceptance length but loses wall-clock speedup (Table 10): averaged across the four datasets, α rises from 7.62 to 8.24 ($+8\%$) while speedup regresses from $5.35\times$ to $5.13\times$ (-4%). The asymmetry is structural: the third pass operates on the residual probability mass the second did not already cover, so its marginal acceptance contribution (0.62 tokens) is only about a third of the second pass's gain over single-branch DFlash (1.87 tokens). The extra VP-Drafter forward and wider verification outweigh this marginal recovery, motivating a single re-anchoring stage in the tested setting.

4.4 Rejection-boundary prediction

We test the score on DFlash's first branch using Qwen3-8B, $\gamma = 16$, greedy decoding, and 128 examples per dataset on B300 GPUs. Top-four choices contain the actual boundary in 77.5% of GSM8K and 80.5% of MATH-500 rejection cycles, versus 36.1% and 37.8% for top one (Table 11); full blocks are excluded. Appendix D gives the dataset-specific output caps, per-length counts, and conditional-confidence analysis.

4.5 Concurrency and the cost of recovery

We evaluate 128 MATH-500 requests on eight B300 SXM6 GPUs, fixing Qwen3-8B, $\gamma = 16$, greedy decoding, and a 256-token output cap. Each concurrent request has its own stream and KV cache, with shared weights. Table 4 and Figure 3 report throughput across concurrency levels and branch budgets. With $K = 4$, throughput improves over DFlash by $1.28\times$ at concurrency one and $1.14\times$ at concurrency sixteen. Throughput decreases with contention for all methods; increasing K from four to eight gives only about 3% more throughput at concurrency sixteen.

Writing C for mean complete decoding-cycle cost, recovery is beneficial in the steady-state latency model precisely when

$$\frac{\alpha_{\mathrm{D2}}}{\alpha_{\mathrm{DF}}} > \frac{C_{\mathrm{D2}}}{C_{\mathrm{DF}}}. \tag{8}$$

A separate GSM8K stage profile shows that verification's share of decode time rises from 57.6% to 67.8% as concurrency increases from one to eight; the throughput advantage narrows from 13.5% to 4.2%. Appendix E provides the independent-stream protocol and stage timings. H200 and B300 results are reported separately.

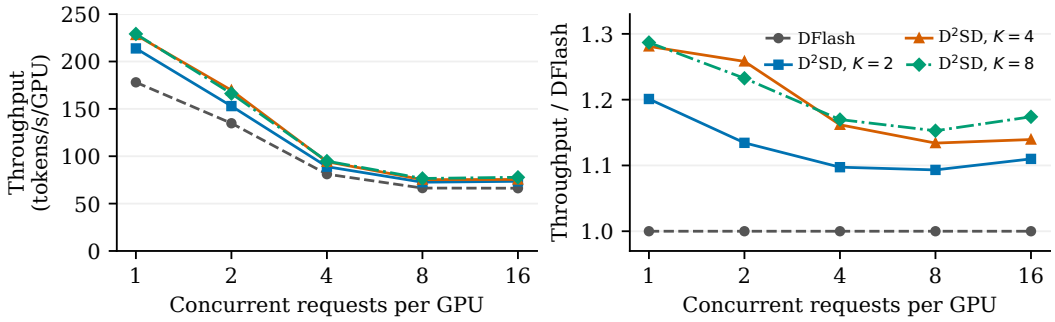


Figure 3: MATH-500 concurrency sweep from Table 4: absolute throughput (left) and throughput relative to DFlash at the same concurrency (right). Points use the same 128 requests per configuration; lines connect the measured settings.

5 Related Work

Speculative Decoding. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) uses a lightweight drafter and target verification to accelerate autoregressive generation while preserving the target distribution. Medusa (Cai et al., 2024) attaches multiple prediction heads to the target; the EAGLE series (Li et al., 2024a;b; 2025b) instead conditions drafting on target hidden states. Complementary approaches use Jacobi trajectories (Fu et al., 2024) or online drafter adaptation (Liu et al., 2023). Tree-based verification amortizes target computation across candidates: SpecInfer (Miao et al., 2024) shares prefix attention, Sequoia (Chen et al., 2024) optimizes tree topology under a verification budget, and EAGLE-2 (Li et al., 2024b) adapts the tree to context. Medusa-2 likewise verifies multi-head candidates with tree attention.

Diffusion Language Models. Diffusion language models generate text by denoising masked sequences, allowing parallel token prediction. LLaDA-1.5 (Zhu et al., 2025) improves alignment through variance-reduced preference optimization, while LLaDA-2.1 (Bie et al., 2026) accelerates sampling with token editing. As speculative drafters, these models combine parallel generation with an autoregressive verifier that preserves the target distribution.

Speculative Decoding with Diffusion Draft Models. DiffuSpec (Li et al., 2025a) uses a large pretrained diffusion drafter, trading longer acceptance for memory and latency overhead. PARD (An et al., 2025) approximates parallel drafting with a small autoregressive model. DFlash (Chen et al., 2026), our first-stage drafter, uses lightweight block diffusion conditioned on target features via KV injection, but a single draft remains vulnerable to early rejection. Concurrent DDTree (Ringel & Romano, 2026) constructs block-diffusion trees using a best-first heap under a fixed node budget. D^{2}SD instead predicts likely rejection boundaries and regenerates their suffixes with a separately trained variable-prefix drafter.

6 Conclusion

We presented D^{2}SD, a dual-diffusion-draft framework that converts a single block draft into shared-prefix candidates. First-draft confidences select likely rejection boundaries; a variable-prefix drafter regenerates their suffixes in one batched pass, and the target jointly verifies all candidates. The results support confidence-guided recovery over block enlargement or naive resampling. Ablations favor a single re-anchoring stage, while concurrency results highlight the cost of recovering additional tokens under contention.

AI USE STATEMENT

Generative AI assisted with manuscript restructuring and editing, LaTeX preparation, literature checking, and scripts that aggregate existing experiment records and produce tables and figures. The assistance also supported metric definitions and the verification explanation. The empirical results originate from the recorded experiments.

REPRODUCIBILITY STATEMENT

Section 3 specifies the boundary score, branch construction, training objective, and verification rule. Section 4 and Appendices A–E describe training budgets, evaluation settings, and metric definitions. The accompanying source contains the numerical table data and scripts for reproducing the aggregation and figures. Appendix F gives the distributional correctness argument.

ETHICS STATEMENT

This work studies inference efficiency using existing language models and benchmarks. The underlying models retain their existing capabilities and risks, including bias and factual errors. Evaluation considers inference efficiency, output behavior, and resource costs.

REFERENCES

Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. Pard: Accelerating llm inference with low-cost parallel draft model adaptation, 2025. URL https://arxiv.org/abs/2504.18583.

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

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, et al. Llada2. 1: Speeding up text diffusion via token editing. *arXiv preprint arXiv:2602.08676*, 2026.

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. In *International Conference on Machine Learning*, pp. 5209–5235. PMLR, 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.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, 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*, 2024.

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

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *12th International Conference on Learning Representations, ICLR 2024*, 2024.

Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, et al. Glide with a cape: A low-hassle method to accelerate speculative decoding. In *International Conference on Machine Learning*, pp. 11704–11720. PMLR, 2024.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf.

Naman Jain, 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. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/94074dd5a072d28ff75a76dabed43767-Paper-Conference.pdf.

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

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

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*, pp. 28935–28948, 2024a.

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, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

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

Mathematical Association of America. American Invitational Mathematics Examination – AIME 2025. https://maa.org/maa-invitational-competitions, February 2025. Accessed: 2026-05-06.

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*, pp. 932–949, 2024.

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

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *International Conference on Learning Representations (ICLR)*, 2025.

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. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

Tengyu Xu, Eryk Helenowski, Karthik Abinav Sankararaman, Di Jin, Kaiyan Peng, Eric Han, Shaoliang Nie, Chen Zhu, Hejia Zhang, Wenxuan Zhou, et al. The perfect blend: Redefining rlhf with mixture of judges. *arXiv preprint arXiv:2409.20370*, 2024.

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

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

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

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, et al. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025.

A TRAINING DETAILS AND COMPUTE ACCOUNTING

For the controlled Qwen3-8B runs, each of DFlash, VP-Drafter, and EAGLE-3 is trained independently for 266,000 optimizer updates. Eight FSDP ranks with four samples per rank give a global batch of 32 and 8,512,000 example presentations across training. Training stops at the specified update count, with gradient accumulation set to one. DFlash and VP-Drafter use maximum context length 3072 and 512 sampled anchors, each expanded into a block of 16 positions. EAGLE-3 uses maximum length 4096 and training-time-test length seven. All use the complete PerfectBlend training source. The target supplies frozen teacher features; the EAGLE-3 objective also uses target token probabilities.

The VP-Drafter uses the normalized truncated prior in Equation 6, with $\beta = 0.9$, and distance decay $\tau = 16$ in Equation 7. The anchor itself is visible and excluded from the masked-token loss. Longer visible prefixes reduce the number of supervised masked positions; dense-equivalent counts below cover the processed positions. DFlash and VP-Drafter share an architecture and specialize through their fixed-anchor and variable-prefix training distributions, respectively.

Table 5: Analytical draft-training estimates for the controlled Qwen3 setup using the dense-equivalent $6NT$ model. N denotes draft parameter count.

Drafter	Parameters N	Dense-equivalent positions/sample	Proxy (EFLOPs)
DFlash	1,048,626,432	$512 \cdot 16 = 8192$	438.7
VP-Drafter	1,048,626,432	$512 \cdot 16 = 8192$	438.7
EAGLE-3	1,021,853,696	$7 \cdot 4096 = 28672$	1496.3

The analytical estimate in Table 5 is

$$F_{\text{draft}} \simeq 6NT_{\text{eq}}, \qquad T_{\text{eq}} = (266000 \cdot 32)\, n_{\text{eq}}. \tag{9}$$

The factor six approximates a dense forward and backward pass. This model accounts for parameter-dependent computation; attention, masking, recomputation, and optimizer overhead depend on the implementation. Adding the 3072-token context as a dense equivalent gives $n_{\text{eq}} = 11264$ and approximately 603.2 EFLOPs per diffusion drafter. Summing the two diffusion drafters gives D^{2}SD a draft-only estimate of 877.4 EFLOPs under the first convention, versus 438.7 for DFlash alone.

Teacher computation is accounted for separately. With the reported Qwen3-8B parameter count $N_t = 8{,}190{,}735{,}360$, the dense forward estimate $2N_tT$ for 8,512,000 sequences of length 3072 is 1284.8 EFLOPs; length 4096 gives 1713.0 EFLOPs. These values apply per stream of training examples. Across two draft-training runs, total teacher cost depends on feature reuse and recomputation. The frozen teacher contributes forward computation, while backward computation is confined to the drafter.

B FULL TARGET-MODEL RESULTS

Tables 1 and 2 give the full target-model results in the main text. Per-task values and averages are rounded separately. The H200 benchmarks and B300 diagnostic and concurrency experiments are reported by hardware and experimental configuration.

For GPT-OSS-20B, DFlash and D^{2}SD use the public first-draft checkpoint `https://huggingface.co/z-lab/gpt-oss-20b-DFlash`. At $T = 0$, D^{2}SD improves both metrics on every task relative to DFlash. At $T = 1$, average speedup rises from $1.71\times$ to $1.84\times$ and acceptance from 1.99 to 2.35. The table reports both metrics for each task, capturing the interaction between acceptance and per-cycle cost.

Table 6: Qwen3-8B block-size control. Each row uses its own decay-matched training schedule. TPF denotes tokens per forward pass in this experiment.

Block size γ	MATH-500 TPF	GSM8K TPF
8	5.04	5.00
16	6.05	5.95
24	6.01	6.00
32	5.85	5.93

C Block size and per-task ablations

C.1 Increasing the block size

Table 6 motivates reallocating the draft budget across branches. Each block size uses a separately trained, decay-matched checkpoint. Table 1 and the concurrency sweep use the default $\gamma = 16$ configuration.

Table 7: TPF on GSM8K (Qwen3-8B, γ=16, K=4): adding K naive resampled branches to DFlash lifts TPF by only 0.37, while D^{2}SD raises it by 2.25 under the same branching budget; full multi-dataset ablation in Section 4.3.

Method	GSM8K TPF
DFlash	6.96
+ K naive samples	7.33
D^{2}SD	**9.21**

C.2 Branch source and cascade depth

Table 8: Augmenting argmax DFlash with K=4 resampled branches yields only modest gains over single-branch DFlash, while D^{2}SD improves both metrics substantially under the same branching budget. Greedy target decoding (T=0); each cell is speedup × / mean acceptance length α.

Dataset	DFlash	$+K$ samples	D^{2}SD
GSM8K	5.51×/6.96	6.18×/7.33	**6.54×/9.21**
MATH	5.11×/6.55	5.67×/6.82	**6.03×/8.56**
MBPP	4.07×/5.09	4.66×/5.47	**5.03×/6.96**
MT-Bench	3.23×/4.41	3.74×/4.70	**3.80×/5.75**
Avg.	4.48×/5.75	5.06×/6.08	**5.35×/7.62**

The resampling control retains the argmax first branch and adds four per-position multinomial samples at draft temperature one; the target remains greedy. The DFlash→DFlash control retains the same confidence-guided prefix selection and joint verification as D^{2}SD, changing the second model only. The third-level control selects one additional boundary per second-stage branch and adds up to four more candidates. Together, these controls examine branch source, variable-prefix training, and cascade depth.

Table 9: Reusing DFlash as the second drafter inside D^{2}SD's cascade pipeline (K=4). Confidence-guided branch placement raises α over single-branch DFlash, but the lack of variable-prefix training keeps both metrics below full D^{2}SD. Each cell is speedup $\times$ / mean acceptance length α.

Dataset	DFlash	DFlash→DFlash	D^{2}SD
GSM8K	5.51×/6.96	5.76×/7.95	**6.54×/9.21**
MATH	5.11×/6.55	5.28×/7.35	**6.03×/8.56**
MBPP	4.07×/5.09	4.36×/5.90	**5.03×/6.96**
MT-Bench	3.23×/4.41	3.35×/4.91	**3.80×/5.75**
Avg.	4.48×/5.75	4.69×/6.53	**5.35×/7.62**

Table 10: Stacking a third VP-Drafter level on top of D^{2}SD (K=4). Every second-draft branch is re-anchored once at its own top-1 predicted rejection boundary, producing up to $2K{+}1 = 9$ shared-prefix candidates per cycle. Mean acceptance length improves modestly, but the extra drafting and verification cost outweighs the marginal recovery, regressing wall-clock speedup. Each cell is speedup $\times$ / mean acceptance length α.

Dataset	D^{2}SD	+ 3rd draft
GSM8K	**6.54**×/9.21	6.25×/**9.89**
MATH	**6.03**×/8.56	5.79×/**9.27**
MBPP	**5.03**×/6.96	4.85×/**7.58**
MT-Bench	**3.80**×/5.75	3.63×/**6.20**
Avg.	**5.35**×/7.62	5.13×/**8.24**

D BOUNDARY DIAGNOSTIC DEFINITIONS AND DETAILED COUNTS

The diagnostic runs the first DFlash branch in isolation. The target is Qwen3-8B, block size is 16, decoding is greedy, thinking mode is disabled, and each dataset contributes 128 examples selected after shuffling with seed zero. The runtime seed is 42. The generation caps are 1024 for GSM8K and 256 for MATH-500. The runtime records B300 SXM6, PyTorch 2.13.0+cu130, and CUDA 13.0. The maximum accepted draft length is 15; a full block has no rejection-boundary label. The per-length hit rate is

$$H_K(l) = \frac{\sum_n \mathbf{1}\{L_n = l,\ l \in S_{K,n}\}}{\sum_n \mathbf{1}\{L_n = l\}}, \qquad l < 15. \tag{10}$$

Overall rates pool the scored cycles. GSM8K has 4683 scored cycles with 1692 top-one and 3628 top-four hits; MATH-500 has 5005 scored cycles with 1893 and 4027 hits. Full-block counts are 698 and 417. Table 12 reports the observed hit rates by accepted length.

For the confidence plot, position k is eligible only when $L_{\text{first}} \geq k - 1$. Within confidence bin I, we compute

$$\widehat{a}(I) = \frac{\sum_{n,k} \mathbf{1}\{c_{n,k} \in I,\ L_n \geq k\}}{\sum_{n,k} \mathbf{1}\{c_{n,k} \in I,\ L_n \geq k-1\}}. \tag{11}$$

The horizontal coordinate is the mean confidence among the denominator's eligible positions. Table 13 provides the counts. Conditioning on survival of the earlier positions aligns this statistic with the local survival approximation used in the method.

Table 11: First-branch boundary prediction on B300. Hit rates use only rejection cycles; full-block rates use all cycles. L_{first} excludes the target bonus token.

Dataset	All cycles	Rejections	Full block	$\mathbb{E}[L_{\text{first}}]$	Top-1	Top-4
GSM8K	5,381	4,683	13.0%	6.19	36.1%	77.5%
MATH-500	5,422	5,005	7.7%	5.02	37.8%	80.5%

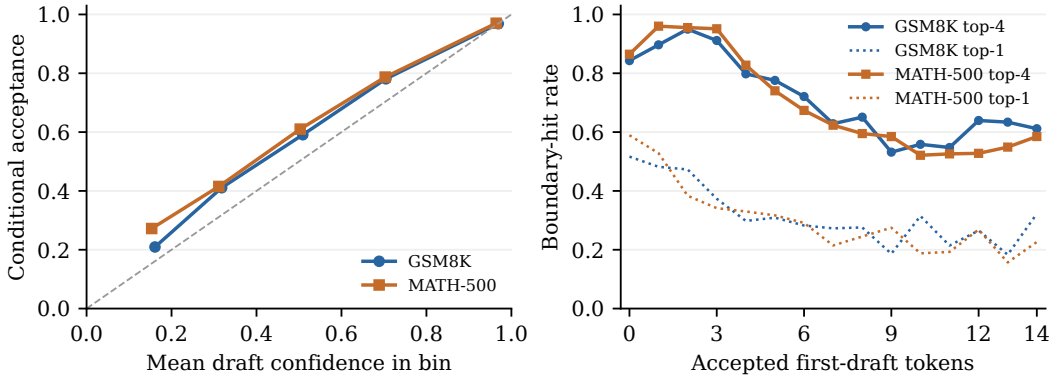


Figure 4: Additional first-branch diagnostics from the recorded B300 cycles. Left: empirical acceptance conditional on all earlier draft tokens surviving, versus mean confidence in five bins. Right: boundary-hit rates by actual accepted draft length. Full-block cycles are excluded from boundary-hit rates. This supplements the original GSM8K plot in Figure 2a.

Table 12: Boundary-hit rates (%) by actual first-branch accepted draft length. Full blocks contribute to mean length but not to hit-rate denominators.

	GSM8K			MATH-500		
L_{first}	Cycles	Top-1	Top-4	Cycles	Top-1	Top-4
0	573	51.7	84.3	784	58.9	86.5
1	639	48.2	89.7	727	52.8	96.0
2	480	47.3	95.0	670	38.4	95.5
3	474	37.3	91.1	533	34.1	95.1
4	406	29.8	79.8	406	33.0	82.8
5	330	30.9	77.6	366	31.7	74.0
6	301	28.2	72.1	285	29.1	67.4
7	253	27.3	62.8	247	21.5	62.3
8	232	27.6	65.1	200	24.5	59.5
9	220	18.6	53.2	171	27.5	58.5
10	197	31.5	55.8	165	18.8	52.1
11	168	21.4	54.8	135	19.3	52.6
12	147	26.5	63.9	108	26.9	52.8
13	142	18.3	63.4	102	15.7	54.9
14	121	32.2	61.2	106	22.6	58.5
15 (full)	698	–	–	417	–	–

Table 13: Conditional confidence bins. Only positions reached by first-branch verification enter the denominator. Acceptance is in percent.

	GSM8K			MATH-500		
Confidence	Positions	Mean c	Accept	Positions	Mean c	Accept
$[0.0, 0.2)$	191	0.161	20.9	463	0.153	27.2
$[0.2, 0.4)$	1731	0.318	41.1	2476	0.311	41.5
$[0.4, 0.6)$	3890	0.509	59.1	4219	0.503	61.0
$[0.6, 0.8)$	4836	0.705	78.1	4522	0.703	78.7
$[0.8, 1.0]$	27335	0.969	96.8	20519	0.964	97.0

E CONCURRENCY PROTOCOL AND STAGE PROFILING

The 128-request MATH-500 sweep uses eight B300 SXM6 devices with 16 requests assigned to each GPU. There is one warmup batch capped at 32 new tokens. The measured runs cap each request at 256 new tokens. The environment records PyTorch 2.13.0+cu130, CUDA 13.0, and Transformers 4.57.3; the experimental configuration uses FlashAttention-2, with FlashInfer for cascade verification. Each request owns an independent stream, caches, and graph runner. Workers have separate Python model state while sharing parameter storage. This independent-stream protocol measures concurrent execution, including host scheduling and synchronization.

Throughput is total generated tokens divided by summed per-GPU request-batch wall time. The latter includes prefill and decode, excludes warmup, and counts each batch until its requests have returned. This gives an aggregate rate per GPU. At $B = 16$, each device processes one measured batch. Branch counts and concurrency values all use the same 128 prompts. Total generated tokens are 32,170 for DFlash and 32,160, 32,177, and 32,222 for $K = 2, 4, 8$. Throughput uses each method's actual token count.

The shorter GSM8K sweep uses 64 shuffled questions, up to 64 new tokens, and a 16-token warmup. It measures both speculative decoding and an autoregressive baseline at the same output lengths. Table 14 reports D^{2}SD stage costs. The matched DFlash stage totals are 31.88 ms at $B = 1$ and 665.71 ms at $B = 8$, compared with 33.99 and 785.46 ms for D^{2}SD. Using $\alpha_{\mathrm{DF}} = 5.30$, the break-even D^{2}SD acceptance is approximately 5.65 and 6.25; measured acceptance is 6.56. DFlash throughput is 140.9 and 50.2 tokens/s/GPU, versus 160.0 and 52.3 for D^{2}SD. Stage totals describe the decoding-cycle cost, while throughput also includes whole-request overhead.

The short $K = 2$ and $K = 8$ GSM8K controls use 32 examples, and the $K = 4$ run uses 64. The main branch-budget comparison therefore uses the uniform 128-example MATH-500 sweep. Results are reported separately for each dataset, output cap, and profiling configuration. Verification share denotes its fraction of measured decoding time.

Table 14: GSM8K, B300, $K = 4$, 64 requests with a 64-token cap. Stage times are CUDA-event milliseconds per request per iteration under contention. Throughput is tokens/s/GPU; the last column is throughput relative to the matched autoregressive run.

B	Draft 1	Draft 2	Verify	Other	Throughput	/ AR
1	5.61	4.75	19.57	4.06	160.0	4.00
2	12.99	12.46	59.69	7.64	112.8	4.15
4	41.70	43.58	222.43	26.03	62.4	4.23
8	94.40	99.42	532.60	59.04	52.3	4.00

F VERIFICATION RULE AND DISTRIBUTIONAL CORRECTNESS

Distributional argument. Let x be the committed context, and let $\mathcal{C}$ contain arbitrary proposed continuations. At a visited prefix z, every surviving candidate has prefix z. Under causal target attention, its precomputed next-token distribution equals $P_t(\cdot \mid x, z)$, independent of its unvisited suffix. Draw one token from this conditional, append it to the output, and retain candidates whose next token matches. If none match, end the cycle and start a new one from the emitted token. Otherwise continue. On reaching the end of a candidate block, draw the bonus token from the target conditional there. Each emitted token thus has the same conditional distribution as autoregressive target sampling. By the chain rule the joint output distribution agrees, up to the same EOS and length stopping rules. Replacing each draw with argmax gives the greedy version. The argument applies to both deterministic and random candidate generation; correctness follows from the target conditionals at the visited prefixes.

This distributional result holds for exact target conditionals with branch-local causal attention and a committed KV cache copied from a surviving path. The prefix walk maintains one set of surviving branches and makes one target draw at each visited prefix before filtering candidates, as specified in Section 3.3.