D3D: DECOUPLED DIFFUSION DRAFTING FOR ASYNCHRONOUS SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Target-conditioned diffusion drafters predict candidate tokens in parallel, yet each new block must wait for updated target-model hidden states. This forces drafting and verification to run sequentially, limiting speedup even though each block is generated in one pass. We introduce D3D, a standalone diffusion drafter that removes this inter-round dependency. D3D combines blockwise denoising with auxiliary autoregressive supervision to achieve long accepted prefixes without target hidden states. This independence lets D3D draft successor blocks during verification in one packed pass, enabling asynchronous speculative decoding. We progressively prune layers according to their effect on acceptance length to reduce the cost of packed drafting. Removing five of 36 layers retains over 99% of mean acceptance length across three benchmarks with Qwen3-8B as the target. Across seven benchmarks, D3D averages 5.00× and 5.71× decoding speedup with Qwen3-8B and Qwen3-32B targets, compared with 4.03× and 3.67× for DFlash. On GSM8K with Qwen3-32B, speedup reaches 7.48×.

1 INTRODUCTION

Autoregressive language models dominate open-ended text generation, but their inherently sequential decoding limits token-level parallelism. Generating one token at a time leads to poor hardware utilization and high inference latency, especially in memory-bound settings. Speculative decoding alleviates this bottleneck by using a lightweight draft model to propose multiple future tokens, which the target model then verifies in parallel (Stern et al., 2018; Xia et al., 2023; Leviathan et al., 2023; Chen et al., 2023; Kim et al., 2023; Miao et al., 2024). With a suitable acceptance rule, speculative decoding preserves the target distribution, yielding the same output distribution as standard decoding with the target model.

The speedup achieved by speculative decoding depends on the number of accepted tokens, draft latency, and verification cost. Recent work has therefore focused on improving draft accuracy while keeping draft latency low. Many methods attach auxiliary predictors to the target model or condition a separate drafter on intermediate representations from the target model (Cai et al., 2024; Ankner et al., 2024; Gloeckle et al., 2024; Li et al., 2024; 2025; Cheng et al., 2024; Xiao et al., 2024). However, autoregressive and recurrent drafters still generate candidate tokens sequentially, causing draft latency to grow with the candidate length.

Parallel drafting addresses the sequential generation cost by predicting multiple candidate positions simultaneously. PARD adapts a pretrained autoregressive model into a parallel drafter, while DiffuSpec and SpecDiff-2 use diffusion models to generate multiple candidate tokens concurrently (An et al., 2025; Li et al., 2026; Sandler et al., 2025). PARD-2 further aligns training with acceptance length and supports both target-dependent and target-independent inference (An et al., 2026). DART and DFlash improve draft quality by conditioning parallel drafters on intermediate representations from the target model (Liu et al., 2026; Chen et al., 2026). In particular, DFlash uses a block diffusion model to generate an entire candidate block in a single forward pass. Recent methods improve target-conditioned parallel drafts by enriching target-feature fusion (Zhang et al., 2026a), refining block predictions with causal context (Huang et al., 2026; Rheinboldt et al., 2026; Zheng & Li, 2026), and weighting training positions according to their contribution to acceptance (Wu et al., 2026). Target-conditioned parallel heads have also been used to generate causal draft trees (Hu et al., 2026). In these implementations, the next draft block requires updated target-model representations.

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.

For a diffusion drafter that generates an entire candidate block in a single forward pass, drafting latency still depends on the drafter's depth. Recent work reports a log-linear relationship between decoder depth and acceptance rate for shallow autoregressive drafters (Yan et al., 2025b). Larger drafters can improve acceptance, but typically require more time to generate each candidate block. When drafting and verification execute sequentially, this additional draft latency can offset the benefit of accepting more tokens. This creates a trade-off between draft quality and draft latency.

Beyond the cost of each forward pass, conditioning on fresh target-model representations creates an execution dependency between drafting rounds. In DFlash, verification produces representations of newly accepted tokens that the drafter consumes in the next round (Chen et al., 2026). The next draft must therefore wait for these representations, even though the drafter predicts all positions within a block in parallel.

Inspired by Speculative Speculative Decoding (SSD), we prepare drafts for possible verification outcomes while the target verifies the current block (Kumar et al., 2026). SSD demonstrates this approach with autoregressive drafters that generate candidate sequences through multiple sequential steps. We use a standalone diffusion drafter to predict each candidate block from its conditioning prefix in a single forward pass. The modeling challenge is to obtain long accepted prefixes without target hidden states while keeping drafting fast enough to complete within the verification interval.

We introduce D3D, a decoupled diffusion drafter built around target-aligned training and progressive layer pruning. Our contributions are as follows:

1. **Target-aligned standalone diffusion drafting.** We combine blockwise denoising and auxiliary autoregressive supervision on target-generated responses to train a standalone, single-pass drafter from pretrained weights. At block size 16, D3D exceeds every baseline in acceptance length on each of the seven benchmarks for both targets, averaging 7.88 with Qwen3-8B and 7.73 with Qwen3-32B (Table 1).
2. **Pruning redundant drafter layers.** We remove layers with the smallest effect on validation acceptance length, with recovery training between removals. With Qwen3-8B as the target, removing five of 36 layers (13.9%) retains over 99% of mean acceptance length and raises mean speedup from $4.44\times$ to $4.69\times$ across three benchmarks (Figure 2).
3. **Asynchronous speculative decoding.** During target verification, D3D prepares possible successor blocks in one packed drafter pass and reuses a matching block in the next round. Across seven benchmarks, it achieves the highest mean decoding speedup over autoregressive decoding among the evaluated methods: $5.00\times$ with Qwen3-8B and $5.71\times$ with Qwen3-32B (Table 1).

2 Related Work

2.1 Parallel and Diffusion Drafting

PARD adapts an autoregressive backbone for masked parallel prediction (An et al., 2025). PARD-2 uses causal masked-token prediction and stochastically injects target features during training, allowing the same drafter to operate with or without these features at inference (An et al., 2026). DFlash injects projected target features into the attention keys and values of each diffusion layer (Chen et al., 2026). D^{2}SD retains this conditioning in both its initial drafter and its second drafter, which generates continuations from variable-length prefixes (Zhang et al., 2026b). D^{2}SD jointly verifies these alternative continuations in the current round. D3D prepares successor blocks for possible next-round prefixes while the target verifies the current block. DeLS-Spec independently trains a short-context predictor and combines its logits with those of a frozen DFlash backbone (Zheng & Li, 2026). Its decoupling separates component training, while the backbone continues to consume target features. D3D shares target-independent inference with PARD and PARD-2, and learns its own prefix representations through diffusion and autoregressive supervision on target-generated responses. Its masked blocks use bidirectional attention, and training does not inject target hidden states. For D3D, we further prune layers based on their measured effect on acceptance length.

SpecDiff-2 derives streak distillation from a surrogate for the expected accepted prefix length (Sandler et al., 2025). PARD-2 weights token losses using the cumulative product of the target model's confidence in preceding tokens (An et al., 2026). D-PACE derives example-dependent position

weights from a surrogate for expected accepted length (Wu et al., 2026). DFlash uses exponentially decaying position weights to emphasize early predictions (Chen et al., 2026). We retain DFlash's position weighting and pair blockwise denoising with auxiliary autoregressive supervision on the same target-generated responses. The auxiliary objective trains next-token prediction from unmasked response prefixes, adding a distinct conditioning pattern to the masked-block training objective.

Other methods optimize how candidate tokens are organized and selected for verification. DiffuSpec searches for causally consistent paths through parallel token predictions and adapts draft length using recent acceptance feedback (Li et al., 2026). SpecDiff-2 selects among multiple sampled drafts using a verifier-based criterion (Sandler et al., 2025). DART and DDTree organize parallel predictions into candidate trees (Liu et al., 2026; Ringel & Romano, 2026), while CaDDTree jointly selects tree structure and node budget using a throughput objective (Zhang et al., 2026c). DSpark couples a parallel backbone with a lightweight sequential module and selects verification length using prefix acceptance estimates and serving load (Cheng et al., 2026). These methods modify candidate dependencies or verification choices; D3D studies the training and layer redundancy of the underlying standalone diffusion drafter.

2.2 Layer Pruning for Efficient Drafting

ShortGPT ranks layers by the similarity between their input and output representations (Men et al., 2025). SDFP uses Fisher information to select layers for removal and constructs drafters without recovery training (Wei et al., 2026). Yan et al. (2025a) study structured pruning for shallow draft architectures and show that language-model accuracy is a poor predictor of speculative-decoding performance. D3D evaluates layer removal through its effect on acceptance length, directly measuring the retained model's utility for block drafting.

Self-data distillation restores quality after layer pruning using responses generated by the unpruned model (Thangarasa et al., 2025). SD^2 applies self-distillation to fine-grained sparse drafters and also evaluates layer-pruned alternatives (Lasby et al., 2025). Our procedure alternates layer removal with recovery training and reassesses acceptance-based layer importance after recovery, so each pruning decision reflects the updated drafter. Within diffusion drafting, DFlare studies greater depth with layer-specific fusion of target features (Zhang et al., 2026a); D3D studies depth reduction while retaining nearly all of a trained standalone drafter's acceptance length.

2.3 Asynchronous Speculative Decoding

PipeInfer maintains speculative execution during pipeline-parallel inference and cancels computation along invalid paths (Butler et al., 2024). SwiftSpec separates drafting and verification across GPU groups and coordinates parallel tree generation with cache management (Zhang et al., 2025). Speculative Speculative Decoding (SSD) caches follow-up drafts for possible verification outcomes and uses a fallback strategy when the observed outcome has no cached draft (Kumar et al., 2026). SSD explicitly accommodates different drafter architectures, including diffusion models. Inspired by SSD, D3D combines target-aligned drafter training and acceptance-guided pruning with packed successor prediction during target verification.

3 Methodology

3.1 Overview

D3D is a standalone diffusion drafter M_{d} for asynchronous speculative decoding with an autoregressive target model M_{t}. Conditioned on the token prefix, M_{d} computes its own prefix representations and generates a candidate block in a single forward pass, without accessing target hidden states. The target verifies the candidate tokens in parallel using rejection sampling and appends one additional token after the accepted prefix.

Our objective is to fully hide drafting latency within target verification while maintaining long accepted prefixes. When drafting is fully hidden, target verification determines the ideal latency of each decoding round. We initialize M_{d} from a pretrained autoregressive model and train on target-generated responses using blockwise denoising and auxiliary autoregressive supervision (Sec-

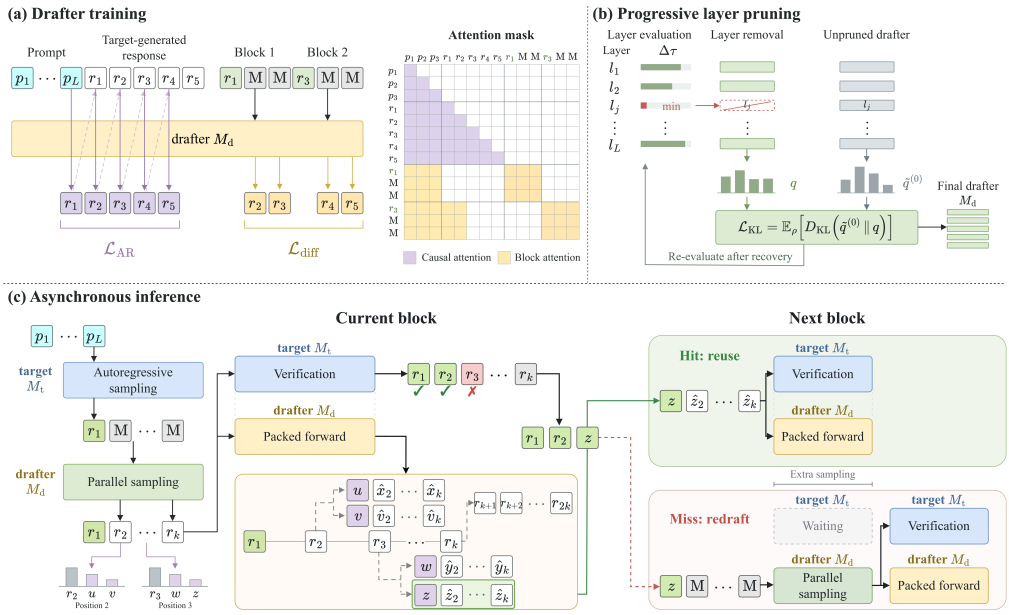

Figure 1: Overview of D3D. **(a) Drafter training.** Target-generated responses supervise blockwise denoising and auxiliary autoregressive prediction. Packed attention shares causal prefix computation while isolating diffusion blocks and allowing bidirectional attention within each block. **(b) Progressive layer pruning.** Each iteration removes the layer causing the smallest decrease in validation acceptance length, followed by recovery and reevaluation of layer importance. Recovery uses distillation from a fixed unpruned drafter whose output distribution is adjusted toward the supervised token. **(c) Asynchronous inference.** During target verification, one packed drafter pass prepares successor blocks for alternative rejection outcomes and full acceptance. A branch matching the verified prefix is reused; otherwise, the drafter generates a new block synchronously.

tion 3.2). We then exploit redundancy across drafter layers through acceptance-guided pruning to reduce drafting latency while preserving the trained drafter's acceptance length (Section 3.3). Section 3.4 describes concurrent drafting and verification and the conditions for hiding drafting latency. Figure 1 summarizes the training, pruning, and inference stages.

3.2 Target-Aligned Drafter Training

A standalone drafter must encode the token prefix and predict continuations using its own representations. We initialize M_d from a pretrained autoregressive model to use its learned language priors. For each prompt $p = (p_1, \ldots, p_L)$, we generate a response $r = (r_1, \ldots, r_T)$ using M_t and train M_d on these response tokens.

We randomly sample K anchor positions

$$\mathcal{S}(r) = \{s_1, \ldots, s_K\}, \qquad 1 \le s_1 < \cdots < s_K < T, \tag{1}$$

and construct a diffusion block of size B for each anchor:

$$\widetilde{B}_k = \big(r_{s_k}, \underbrace{[\mathrm{M}], \ldots, [\mathrm{M}]}_{B-1}\big). \tag{2}$$

Here, $[\mathrm{M}]$ denotes a mask token marking a position to be predicted. We use a reserved token from the pretrained model's vocabulary as $[\mathrm{M}]$. The anchor token r_{s_k} remains visible, and the drafter predicts the $B-1$ masked positions in parallel. For each block, only the next $b_k = \min(B-1, T-s_k)$ response tokens are supervised. Masked positions beyond the response boundary remain in the block and participate in attention but are excluded from the loss.

We pack the K diffusion blocks into a single sequence to obtain denoising supervision for the masked positions following each anchor in one forward pass (Figure 1(a)). M_d computes a shared

causal encoding of (p, r) to avoid redundant prefix computation. Each block $\tilde{B}_k$ attends only to the representations of $(p, r_{<s_k})$ and its own tokens, with bidirectional attention within the block.

For the j-th masked position in the k-th block, we define

$$\ell_{k,j}^{\text{diff}} = -\log P_{M_{\text{d}}}\left(r_{s_k+j} \mid p, r_{<s_k}, \tilde{B}_k\right). \tag{3}$$

Here, $P_{M_{\text{d}}}$ denotes the drafter's conditional token distribution. Verification retains candidate tokens up to the first rejection, motivating greater weight on early block positions. Following DFlash (Chen et al., 2026), we use exponentially decaying position weights:

$$w_j = \exp\left(-\frac{j-1}{\gamma}\right), \quad \text{where } j \in \{1, \ldots, B-1\}. \tag{4}$$

The parameter $\gamma > 0$ controls the decay rate. The position-weighted diffusion objective is

$$\mathcal{L}_{\text{diff}} = \frac{1}{Z_{\text{diff}}} \sum_{k=1}^{K} \sum_{j=1}^{b_k} w_j \ell_{k,j}^{\text{diff}}, \quad \text{where} \quad Z_{\text{diff}} = \sum_{k=1}^{K} \sum_{j=1}^{b_k} w_j. \tag{5}$$

To preserve the pretrained model's language modeling capability, we apply an auxiliary autoregressive objective to the unmasked target-generated response:

$$\mathcal{L}_{\text{AR}} = -\frac{1}{T} \sum_{i=1}^{T} \log P_{M_{\text{d}}}(r_i \mid p, r_{<i}). \tag{6}$$

The complete training objective is

$$\mathcal{L}_{\text{train}} = \mathcal{L}_{\text{diff}} + \lambda \mathcal{L}_{\text{AR}}, \tag{7}$$

where λ controls the contribution of the autoregressive objective.

3.3 Acceptance-Guided Layer Pruning

We exploit redundancy across drafter layers to reduce drafting latency while preserving average acceptance length. With the target model and block size fixed, we measure validation acceptance length after separately removing each remaining layer (Figure 1(b)). We remove the single layer whose removal causes the smallest decrease in average acceptance length, then perform recovery training. Layer removal changes the inputs to subsequent layers, and recovery training updates the remaining parameters. We therefore recompute layer importance on the recovered model before the next pruning step.

Recovery training uses knowledge distillation (Hinton et al., 2015) from the trained, unpruned drafter, which remains fixed throughout pruning. At each supervised prediction position, let y denote the supervised token from the target-generated response, and let $q^{(0)}$ and q denote the teacher and pruned drafter distributions. We compute both distributions with the same block input and attention pattern. Direct distillation can favor a competing token when the teacher's most probable prediction differs from y. We adjust the teacher distribution toward the supervised token y by mixing it with the one-hot distribution e_y:

$$\tilde{q}^{(0)} = (1-\eta)q^{(0)} + \eta e_y. \tag{8}$$

At each prediction position, we choose the smallest mixing weight η that makes y more probable than every other token by at least a fixed positive margin. Choosing the smallest weight retains as much of the teacher's soft distribution as possible. Appendix A provides the closed-form expression for η.

The distillation loss is the position-weighted forward KL divergence from the adjusted teacher distribution to the pruned drafter:

$$\mathcal{L}_{\text{KL}} = \mathbb{E}_{\rho}\left[D_{\text{KL}}\left(\tilde{q}^{(0)} \| q\right)\right]. \tag{9}$$

Here, $\mathbb{E}_{\rho}$ denotes a normalized average over masked prediction positions, with weights that decay exponentially with distance from each block's anchor. We combine this loss with the training objective from Section 3.2:

$$\mathcal{L}_{\text{rec}} = \mathcal{L}_{\text{diff}} + \lambda \mathcal{L}_{\text{AR}} + \mathcal{L}_{\text{KL}}. \tag{10}$$

3.4 INFERENCE

M_{t} generates the first response token as the initial anchor, and M_{d} predicts $B-1$ candidate tokens in a single forward pass, forming a block of size B. The target verifies these candidates in parallel using rejection sampling with their corresponding proposal probabilities (Leviathan et al., 2023). It appends the accepted candidates and samples the next token from the residual distribution at the first rejection, or from the target distribution if all candidates are accepted.

During verification, M_{d} prepares continuations for possible verification outcomes in a single packed forward pass (Figure 1(c)). Each rejection branch extends the existing output prefix with the candidates preceding the rejection position, an alternative anchor, and a predicted continuation. For full acceptance, the drafter also prepares a continuation conditioned on the entire current block.

A cache hit occurs when the complete verified prefix, including the target-generated token, matches the initial token sequence of a prepared branch. We reuse the remaining draft tokens of the matching branch for the next verification round. On a cache miss, the target must wait for M_{d} to synchronously generate a new block from the verified prefix before the next verification round can begin. We define the cache hit rate p_{hit} as the probability of a hit in a verification round.

Position coverage C includes the first $C-1$ rejection positions and the full-acceptance case. Token fan-out F is the number of alternative anchors per selected rejection position (Kumar et al., 2026). We select the F most probable tokens under the drafter distribution, excluding the proposed token; under greedy drafting, these have ranks $2, \ldots, F+1$.

The packed input contains the current block which is being verified, $(C-1)F$ blocks for rejection outcomes, and one block for full acceptance. With B token positions per block, the asynchronous draft budget is

$$N_{\mathrm{async}} = B\left[(C-1)F + 2\right]. \tag{11}$$

Let t_{d} and t_{v} denote the latencies of packed drafting and target verification under concurrent execution. For stages that begin simultaneously, a cache hit hides the packed drafting latency when $t_{\mathrm{d}} \leq t_{\mathrm{v}}$. Larger budgets can increase t_{d} and reduce the latency benefit obtained from a higher cache-hit rate. Acceptance-guided pruning aims to reduce t_{d} with minimal loss in acceptance length.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models and Evaluation We evaluate D3D with Qwen3-8B and Qwen3-32B as target models across seven benchmarks. These cover mathematical reasoning (GSM8K, MATH-500, and AIME25), code generation (HumanEval, MBPP, and LiveCodeBench-v6), and open-ended dialogue (MT-Bench). We report decoding speedup relative to autoregressive decoding and average acceptance length τ. We define τ as the mean number of output tokens added per verification round, including the token generated by the target model.

Training Data We use prompts from Open-PerfectBlend, an instruction-tuning dataset containing 1.42 million examples. We regenerate responses with the corresponding target model in non-thinking mode, using its recommended sampling settings.

Baselines We compare D3D with DFlash (Chen et al., 2026), DSpark (Cheng et al., 2026), EAGLE-3 (Li et al., 2025), PARD-2 (An et al., 2026), and SSD (Kumar et al., 2026). PARD-2 and SSD are evaluated using asynchronous inference. For Qwen3-8B, we use the public DSpark and EAGLE-3 checkpoints released by the DSpark authors, both trained on Open-PerfectBlend. We train DFlash and PARD-2 for both targets, and DSpark and EAGLE-3 for Qwen3-32B, on the same target-generated training data as D3D.

Implementation The main experiments use D3D-8B, initialized from Qwen3-8B and progressively pruned from 36 to 31 Transformer layers, with $B=16$, $C=12$, and $F=2$ for both targets (Section 3.4). Target-aligned drafter training uses 128 random anchors per example and 3 training epochs (360 GPU hours). Target-aligned drafter training and recovery training use BF16 on two

NVIDIA H800 GPUs with 8-bit AdamW. We use a local batch size of 1 and gradient accumulation steps of 8, giving a global batch size of 16. We use a cosine learning-rate schedule from 10^{-5} to 10^{-6} with 1% warmup, zero weight decay, and gradient-norm clipping at 1.

For pruning, we reuse the target-aligned training data and perform 10k recovery training steps per stage. Layer selection and checkpoint selection are performed on two separate, non-overlapping development sets, with acceptance length evaluated on 512 examples per set.

All evaluations use the Transformers backend with greedy decoding in non-thinking mode. D3D, SSD, and PARD-2 use two H800 GPUs; the synchronous DFlash, DSpark, and EAGLE-3 baselines run on one.

4.2 Main Results

Table 1: Speedup and average acceptance length τ across mathematical reasoning, code generation, and chat benchmarks. LCB denotes LiveCodeBench. Numbers in parentheses indicate block sizes.

Model	Method	Math						Code						Chat			
		GSM8K		MATH-500		AIME25		HumanEval		MBPP		LCB-v6		MT-Bench		*Avg.*	
		Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Qwen3 8B	EAGLE-3 (8)	3.43×	5.58	3.34×	5.38	3.01×	4.76	2.95×	4.71	2.71×	4.31	2.88×	4.30	1.87×	2.90	2.88×	4.56
	DFlash (16)	4.59×	7.93	5.00×	7.44	4.34×	6.29	4.34×	6.02	4.03×	5.29	3.48×	5.32	2.46×	3.48	4.03×	5.96
	DSpark (8)	4.93×	6.47	4.86×	6.40	4.79×	5.75	4.70×	5.84	4.19×	5.48	4.44×	5.40	3.17×	3.82	4.44×	5.59
	PARD-2 (16)	6.16×	7.26	5.81×	7.71	4.44×	5.52	3.95×	5.46	3.39×	4.78	2.89×	4.43	1.89×	4.15	4.08×	5.62
	SSD (4)	1.98×	3.43	2.06×	3.53	2.00×	3.44	1.87×	3.26	1.80×	3.11	1.83×	3.17	1.34×	2.79	1.84×	3.25
	D3D (16)	**6.71×**	**9.87**	**6.21×**	**9.84**	**5.11×**	**7.98**	**5.19×**	**7.79**	**4.70×**	**7.31**	**4.44×**	**6.68**	**2.68×**	**5.71**	**5.00×**	**7.88**
Qwen3 32B	EAGLE-3 (8)	4.08×	6.05	4.16×	5.92	3.53×	5.36	3.72×	5.25	3.54×	4.96	2.91×	4.61	2.31×	3.44	3.46×	5.08
	DFlash (16)	4.46×	5.72	4.37×	5.57	3.99×	5.28	3.75×	4.71	3.48×	4.50	2.66×	4.04	3.00×	3.89	3.67×	4.82
	DSpark (8)	4.59×	6.41	4.40×	6.34	3.98×	5.79	4.25×	5.66	4.33×	5.39	3.58×	5.07	2.35×	3.66	3.92×	5.48
	PARD-2 (16)	5.62×	6.89	5.38×	7.64	4.02×	5.27	3.92×	5.14	3.24×	4.55	2.82×	4.33	2.19×	3.86	3.88×	5.38
	SSD (6)	3.84×	4.62	4.05×	4.77	3.97×	4.61	3.56×	4.11	3.32×	3.92	3.65×	4.14	2.35×	3.41	3.54×	4.22
	D3D (16)	**7.48×**	**9.58**	**7.46×**	**9.68**	**5.85×**	**8.18**	**6.22×**	**7.66**	**5.71×**	**7.20**	**4.25×**	**6.13**	**3.01×**	**5.65**	**5.71×**	**7.73**

Decoding performance. D3D achieves higher average decoding speedup than DFlash, DSpark, EAGLE-3, PARD-2, and SSD for both target models (Table 1). With Qwen3-8B, D3D reaches an average speedup of $5.00\times$, compared with $4.44\times$ for DSpark and $4.03\times$ for DFlash. With Qwen3-32B, D3D reaches $5.71\times$, compared with $3.92\times$ and $3.67\times$, respectively. These correspond to improvements of 12.6% and 45.7% over the highest baseline average in our evaluation. The peak speedup reaches $7.48\times$ on GSM8K with Qwen3-32B.

D3D also achieves longer accepted sequences than all five baselines on every benchmark for both targets. Its average τ is 7.88 with Qwen3-8B and 7.73 with Qwen3-32B, compared with DFlash's 5.96 and 4.82 at the same block size. These results show that conditioning on the token prefix alone supports high acceptance across both target sizes. The speedup advantage varies by benchmark: D3D is fastest on five of seven benchmarks with Qwen3-8B and all seven with Qwen3-32B.

Concurrent serving. Using the Transformers backend with SDPA, we measure aggregate decoding throughput with D3D-1.7B and Qwen3-32B across 1–16 concurrent requests (Table 2). At 16 requests, MATH-500 reaches 864 tokens/s ($2.7\times$), while MT-Bench reaches 386 tokens/s ($1.5\times$). Throughput rises with concurrency on all three tasks, while relative speedup declines. Single-request speedup therefore does not capture how the gain changes under load.

Table 2: Decoding throughput in tokens/s (speedup in parentheses) versus concurrency; D3D-1.7B drafter, Qwen3-32B target.

Requests	MATH-500	HumanEval	MT-Bench
1	154 (6.1×)	135 (5.3×)	75 (2.9×)
4	470 (4.8×)	407 (4.3×)	202 (2.1×)
8	745 (3.9×)	655 (3.9×)	327 (1.9×)
16	864 (2.7×)	748 (2.6×)	386 (1.5×)

4.3 Ablation Studies

Auxiliary autoregressive supervision. We assess the contribution of auxiliary AR supervision by varying λ in Eq. (7) and training each D3D-1.7B variant for one epoch (Table 3, left). Setting $\lambda = 1$

increases τ by 0.24 to 0.40 tokens over diffusion-only training ($\lambda = 0$) across all four benchmarks; increasing λ to 1.5 provides no further improvement. The GSM8K training curves show that the benefit persists throughout training (Appendix B.1). Blockwise denoising trains parallel predictions, while the AR objective continues to supervise next-token prediction from unmasked prefixes. This supervision helps the standalone drafter learn to predict from its own prefix representations without target hidden states. We therefore use $\lambda = 1$ during target-aligned drafter training.

Drafter scale. The scale ablation examines whether D3D's acceptance advantage requires a larger parameter budget. We train its 1.7B, 4B, and 8B variants for one epoch and compare them with DFlash (Table 3, right). D3D-1.7B achieves higher acceptance than DFlash on all three benchmarks with fewer total parameters (2.0B versus 2.3B). On MATH-500, increasing drafter size from 1.7B to 4B improves τ by 1.29 tokens, while increasing it from 4B to 8B adds 0.44 tokens; HumanEval and MT-Bench show similar diminishing gains. The AR and initialization ablations separately show that both training choices improve acceptance at a fixed architecture. The smaller gains at larger scales motivate pruning the trained 8B drafter to reduce drafting cost while retaining its acceptance.

Table 3: Effects of auxiliary AR weight (a) and drafter scale (b) on average acceptance length τ. We evaluate D3D variants after one epoch of training, with Qwen3-8B as the target and $B = 16$.

(a) Auxiliary AR weight

λ	GSM8K	MATH-500	HumanEval	MBPP
0	7.89	7.86	5.86	5.53
0.5	8.05	7.98	6.02	5.64
1.0	**8.29**	**8.16**	**6.25**	**5.77**
1.5	8.25	8.13	6.19	5.75

(b) Drafter scale

Drafter	Params.	MATH-500	HumanEval	MT-Bench
DFlash	2.3B	6.87	5.42	3.23
D3D-1.7B	2.0B	8.16	6.25	4.57
D3D-4B	4.4B	9.45	7.42	5.38
D3D-8B	8.2B	**9.89**	**7.86**	**5.73**

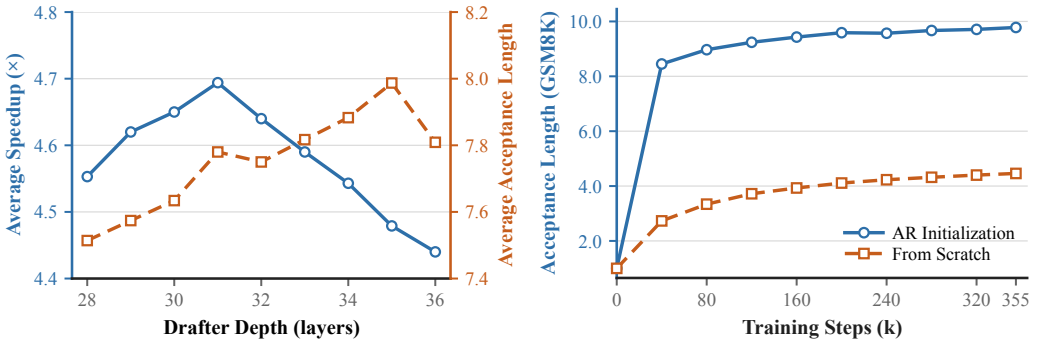

Figure 2: Ablations of D3D-8B depth and initialization with Qwen3-8B as the target and $B = 16$. Left: decoding speedup and acceptance length τ, averaged over MATH-500, HumanEval, and MT-Bench. Right: GSM8K τ during training from pretrained or random initialization.

Acceptance-guided layer pruning. To test whether depth reduction improves speedup with little loss in acceptance, we evaluate progressively pruned and recovered D3D-8B checkpoints under asynchronous inference (Figure 2, left). Pruning from 36 to 31 layers reduces depth by 13.9% while retaining over 99% of mean τ across MATH-500, HumanEval, and MT-Bench. The 31-layer checkpoint achieves the highest mean decoding speedup among the tested depths, at $4.69\times$. The small acceptance loss indicates that part of the original depth is redundant for drafting after recovery. Reducing drafting work makes it easier to overlap drafting with verification, while the retained acceptance keeps the average number of output tokens per round nearly unchanged. Below 31 layers, acceptance length and speedup both decline, indicating that further computational savings no longer compensate for the loss in draft quality. We therefore select the final recovered checkpoint by validation decoding speedup.

Pretrained initialization. Holding the drafter architecture and training budget fixed, we compare Qwen3-8B weights with random initialization to isolate the contribution of pretraining (Figure 2, right). At the final measured training step, pretrained initialization yields $\tau \approx 9.8$ on GSM8K, compared with $\tau \approx 4.5$ for random initialization; the advantage emerges early and persists throughout

training. A pretrained model already predicts plausible continuations from a token prefix, allowing adaptation to focus on target alignment and parallel block prediction. A randomly initialized drafter must also learn this prediction capability within the same training budget.

Asynchronous draft budget. Two sweeps examine the tradeoff between speculative coverage and drafting cost: varying position coverage C at fixed $F = 2$, and varying token fan-out F at fixed $C = 12$ (Figure 3). We use the 31-layer D3D-8B drafter on AIME25. Both sweeps achieve their highest speedup of approximately $5.0\times$ at $C = 12$, $F = 2$, and a 384-token asynchronous draft budget. Increasing C from 12 to 16 at $F = 2$ raises the hit rate from 71.3% to 78.3% but reduces speedup. At $C = 12$, increasing F from 2 to 5 raises the hit rate to 81.8% and the budget to 912 tokens, with lower speedup than at $F = 2$. Cache hits avoid a new drafting pass after verification, reducing the delay before the next round. Increasing coverage adds computation to every packed drafting pass, including branches that verification never selects. We therefore select the budget by decoding speedup to account for cache reuse and the cost of preparing successor blocks.

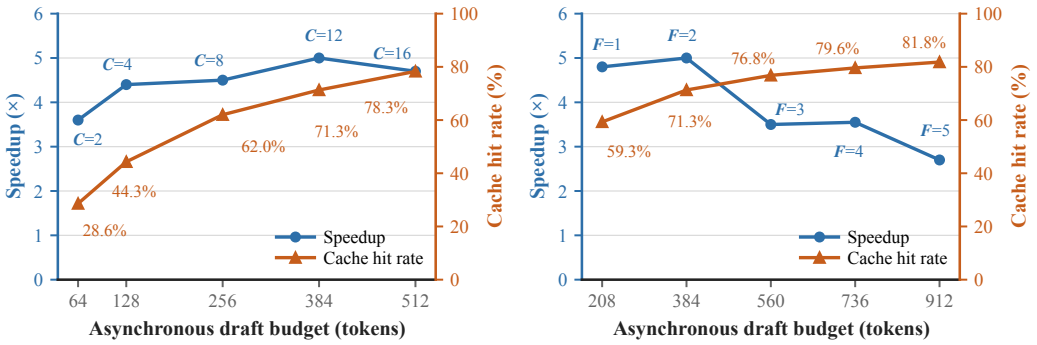

Figure 3: Decoding speedup and cache hit rate versus asynchronous draft budget on AIME25, using the 31-layer D3D-8B drafter with Qwen3-8B and $B = 16$. Left: varying position coverage C at $F = 2$. Right: varying token fan-out F at $C = 12$.

5 Conclusion

We presented D3D, which provides the first empirical demonstration of asynchronous lossless speculative decoding with a standalone diffusion drafter. Target-aligned training combines blockwise denoising with auxiliary autoregressive supervision to adapt a pretrained model for single-pass block prediction without target hidden states. Acceptance-guided progressive pruning with recovery training reduces drafting latency with little loss in acceptance length. Across seven benchmarks, D3D achieves average decoding speedups of $5.00\times$ and $5.71\times$ over autoregressive decoding with Qwen3-8B and Qwen3-32B as targets, respectively. These average speedups exceed those of both SSD and asynchronously deployed PARD-2 in our evaluation. With Qwen3-8B as the target, pruning D3D-8B from 36 to 31 layers retains over 99% of the unpruned drafter's mean acceptance length across three benchmarks. These results show that target-aligned training and acceptance-guided pruning make standalone diffusion drafting effective for asynchronous lossless speculative decoding.

Limitations

We evaluate D3D, SSD, and PARD-2 with one GPU for the target and another for the drafter. The drafter adds model memory and speculative computation. We have not compared throughput per GPU against alternative uses of the same two-GPU budget, such as target inference using tensor parallelism. Cache misses require synchronous drafting before verification can resume. Our implementation uses Transformers; evaluating D3D in production frameworks such as SGLang or vLLM with their batching and scheduling optimizations remains future work.

AI USE STATEMENT

We used generative AI tools to assist with manuscript writing and language polishing, including improving grammar, clarity, and readability, and to help debug code. All AI-assisted text and code suggestions were reviewed and revised by the authors to ensure accuracy and consistency with the reported methods and results. The authors take full responsibility for the content of this work.

REPRODUCIBILITY STATEMENT

We provide the complete training and evaluation code as an anonymized ZIP archive in the supplementary material. Sections 3 and 4.1 describe the method and experimental setup, respectively.

REFERENCES

Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. PARD: Accelerating LLM Inference with Low-Cost PARallel Draft Model Adaptation, November 2025.

Zihao An, Taichi Liu, Ziqiong Liu, Dong Li, Ruofeng Liu, and Emad Barsoum. PARD-2: Target-aligned parallel draft model for dual-mode speculative decoding. *arXiv preprint arXiv:2605.08632*, 2026. doi: 10.48550/arXiv.2605.08632. URL https://arxiv.org/abs/2605.08632.

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-Dependent Draft Heads for Medusa Decoding, October 2024.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024. URL https://aclanthology.org/2024.acl-long.172/.

Branden Butler, Sixing Yu, Arya Mazaheri, and Ali Jannesari. Pipeinfer: Accelerating llm inference using asynchronous pipelined speculation. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–19. IEEE, 2024.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads, June 2024.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating Large Language Model Decoding with Speculative Sampling, February 2023.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding, May 2026.

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, July 2026.

Yunfei Cheng, Aonan Zhang, Xuanyu Zhang, Chong Wang, and Yi Wang. Recurrent Drafter for Fast Speculative Decoding in Large Language Models, December 2024.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & Faster Large Language Models via Multi-token Prediction, April 2024.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Lanxiang Hu, Zhaoxiang Feng, Yulun Wu, Haoran Yuan, Yujie Zhao, Yu-Yang Qian, Bojun Wang, Peng Zhao, Daxin Jiang, Yibo Zhu, Tajana Rosing, and Hao Zhang. JetSpec: Breaking the scaling ceiling of speculative decoding with parallel tree drafting. *arXiv preprint arXiv:2606.18394*, 2026. URL https://arxiv.org/abs/2606.18394.

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.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder. *Advances in Neural Information Processing Systems*, 36:39236–39256, 2023.

Tanishq Kumar, Tri Dao, and Avner May. Speculative Speculative Decoding, May 2026.

Mike Lasby, Nish Sinnadurai, Valavan Manohararajah, Sean Lie, Yani Ioannou, and Vithursan Thangarasa. SD^2: Self-distilled sparse drafters, 2025. URL https://arxiv.org/abs/2504.08838.

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. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 20896–20910, 2026.

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, 2024. URL https://proceedings.mlr.press/v235/li24bt.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *Advances in Neural Information Processing Systems*, 38:136737–136756, 2025.

Fuliang Liu, Xue Li, Ketai Zhao, Yinxi Gao, Ziyan Zhou, Zhonghui Zhang, Zhibin Wang, Wanchun Dou, Sheng Zhong, and Chen Tian. DART: Diffusion-Inspired Speculative Decoding for Fast LLM Inference, January 2026.

Jingyu Liu, Beidi Chen, and Ce Zhang. Speculative prefill: Turbocharging TTFT with lightweight and training-free token importance estimation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 38188–38209, 2025. URL https://proceedings.mlr.press/v267/liu25g.html.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. ShortGPT: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20192–20204. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.1035. URL https://aclanthology.org/2025.findings-acl.1035/.

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*, pp. 932–949, La Jolla CA USA, April 2024. ACM. ISBN 979-8-4007-0386-7. doi: 10.1145/3620666.3651335.

Peer Rheinboldt, Frédéric Berdoz, and Roger Wattenhofer. TreeFlash: Parallel AR-approximation for faster speculative decoding. *arXiv preprint arXiv:2606.03819*, 2026. URL https://arxiv.org/abs/2606.03819.

Liran Ringel and Yaniv Romano. Accelerating Speculative Decoding with Block Diffusion Draft Trees, April 2026.

Jameson Sandler, Jacob K. Christopher, Thomas Hartvigsen, and Ferdinando Fioretto. SpecDiff-2: Scaling Diffusion Drafter Alignment For Faster Speculative Decoding, November 2025.

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

Vithursan Thangarasa, Ganesh Venkatesh, Mike Lasby, Nish Sinnadurai, and Sean Lie. Self-data distillation for recovering quality in pruned large language models. In *Proceedings of Machine Learning and Systems*, volume 7. MLSys, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/af2d9fb5bcee19ef2dfa70d843520c97-Abstract-Conference.html.

Hanyu Wei, Zunhai Su, Peng Lu, Chao Li, Spandan Tiwari, Ashish Sirasao, and Yuhan Dong. SDFP: Speculative decoding with FIT-pruned models for training-free and plug-and-play LLM acceleration, 2026. URL https://arxiv.org/abs/2602.05499.

Tianyu Wu, Yu Yao, Zhenting Qi, Han Zheng, Zhuohan Wang, Haoran Ma, Lawrence Liao, Himabindu Lakkaraju, Ju Li, and Yilun Du. D-PACE: Dynamic position-aware cross-entropy for parallel speculative drafting. *arXiv preprint arXiv:2605.18810*, 2026. URL https://arxiv.org/abs/2605.18810.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3909–3925, 2023.

Bin Xiao, Chunan Shi, Xiaonan Nie, Fan Yang, Xiangwei Deng, Lei Su, Weipeng Chen, and Bin Cui. Clover: Regressive Lightweight Speculative Decoding with Sequential Knowledge, May 2024.

Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. Decoding speculative decoding. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6460–6473. Association for Computational Linguistics, 2025a. doi: 10.18653/v1/2025.naacl-long.328. URL https://aclanthology.org/2025.naacl-long.328/.

Siyuan Yan, Mo Zhu, Guo-qing Jiang, Jianfei Wang, Jiaxing Chen, Wentai Zhang, Xiang Liao, Xiao Cui, Chen Zhang, Zhuoran Song, and Ran Zhu. Scaling Laws for Speculative Decoding, May 2025b.

Jiebin Zhang, Zhenghan Yu, Song Liu, Eugene J. Yu, Zheng Li, Dawei Zhu, Jiangshan Duo, Weimin Xiong, Yifan Song, Guanghua Yu, Jianchen Zhu, and Sujian Li. DFlare: Scaling Up Draft Capacity for Block Diffusion Speculative Decoding, June 2026a.

Liyuan Zhang, Jiarui Zhang, Jinwei Yao, Ran Yan, Yuchen Yang, Jiahao Zhang, Tongkai Yang, Yi Wu, and Binhang Yuan. D^2SD: Accelerating speculative decoding with dual diffusion draft models. *arXiv preprint arXiv:2606.04446*, 2026b. doi: 10.48550/arXiv.2606.04446. URL https://arxiv.org/abs/2606.04446.

Shuai Zhang, Huachuan Qiu, Hongliang He, and Yong Dai. Cost-Aware Diffusion Draft Trees for Speculative Decoding, June 2026c.

Ziyi Zhang, Ziheng Jiang, Chengquan Jiang, Menghan Yu, Size Zheng, Haibin Lin, Henry Hoffmann, and Xin Liu. SwiftSpec: Ultra-low latency LLM decoding by scaling asynchronous speculative decoding, 2025. URL https://arxiv.org/abs/2506.11309.

Hong-Kai Zheng and Piji Li. DeLS-Spec: Decoupled long-short contexts for parallel speculative drafting. *arXiv preprint arXiv:2607.07409*, 2026. doi: 10.48550/arXiv.2607.07409. URL https://arxiv.org/abs/2607.07409.

A Teacher Distribution Adjustment

At each masked prediction position, let $q^{(0)}$ denote the fixed teacher's token distribution and y the supervised token from the target-generated response. The teacher's probability margin for y is

$$m = q^{(0)}(y) - \max_{v \neq y} q^{(0)}(v). \tag{12}$$

The mixture in Section 3.3 changes this margin to $m + \eta(1 - m)$. For a required margin $\delta \in (0, 1)$, the smallest mixing coefficient that satisfies $m + \eta(1 - m) \geq \delta$ is

$$\eta = \max\left(0, \frac{\delta - m}{1 - m}\right). \tag{13}$$

When $m \geq \delta$, $\eta = 0$ leaves the teacher distribution unchanged. Otherwise, the adjustment gives y a margin of δ while retaining the largest possible teacher contribution $1 - \eta$. The mixture scales all non-target probabilities by $1 - \eta$, preserving their ratios. We use $\delta = 0.01$ in our experiments.

B Additional Ablations

B.1 Training Dynamics

We examine whether auxiliary AR supervision provides a sustained benefit by tracking GSM8K acceptance length during one-epoch D3D-1.7B training (Figure 4). After the initial training phase, nonzero AR weights yield higher acceptance than diffusion-only training ($\lambda = 0$). Among the tested weights, $\lambda = 1$ generally gives the highest acceptance; increasing it to 1.5 provides no further benefit. The sustained advantage supports maintaining next-token supervision during diffusion adaptation, where it continues to constrain predictions from unmasked prefixes.

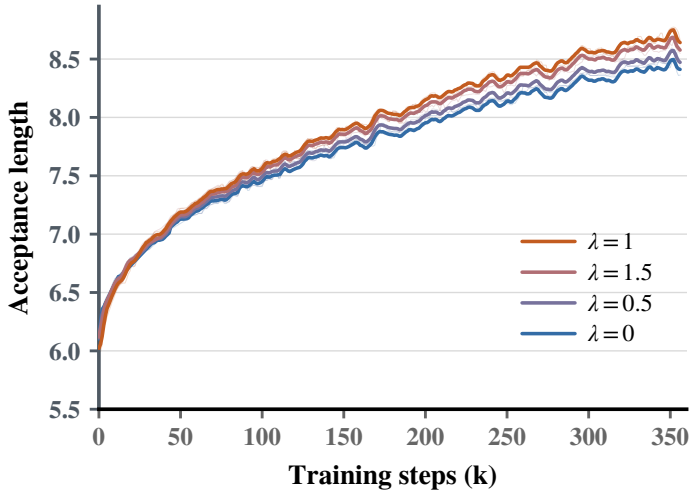

Figure 4: GSM8K acceptance length τ during D3D-1.7B training with different auxiliary autoregressive loss weights λ. The target is Qwen3-8B and the block size is $B = 16$.

B.2 Block Size

To examine how block size affects acceptance and decoding speedup, we train separate D3D-8B drafters for one epoch at $B \in \{8, 16, 32\}$. We evaluate each drafter at its training block size under asynchronous inference with Qwen3-8B (Table 4). Increasing B from 8 to 16 improves speedup on MATH-500 and HumanEval, while MT-Bench is slightly faster at $B = 8$. Increasing B further to 32 raises acceptance length but reduces speedup on all three tasks; on MATH-500, τ increases from 9.89 to 12.39 while speedup falls from $5.55\times$ to $3.16\times$. Larger blocks permit longer accepted prefixes but also add candidate positions for drafting and verification, including tokens that verification may discard. We use $B = 16$ as the default because it achieves the highest mean speedup across the three tasks.

Table 4: Decoding speedup and acceptance length τ versus block size; D3D-8B, Qwen3-8B target.

B	MATH-500		HumanEval		MT-Bench	
	Speedup	τ	Speedup	τ	Speedup	τ
8	4.58×	6.26	4.00×	5.52	2.55×	4.38
16	5.55×	9.89	4.47×	7.85	2.47×	5.72
32	3.16×	**12.39**	2.56×	**8.70**	1.42×	**6.33**

B.3 Number of Training Anchors

The number of sampled anchors controls how many diffusion blocks each response contributes to training. We test whether denser supervision improves acceptance by training D3D-1.7B for one epoch with $K \in \{16, 64, 128, 256\}$ anchors per response, using Qwen3-8B as the target and $B = 16$ (Table 5). Increasing K from 16 to 64 improves τ by 0.25, 0.21, and 0.04 tokens on MATH-500, HumanEval, and MT-Bench, respectively. Increasing K from 64 to 256 adds only 0.03, 0.03, and 0.06 tokens. Given that target-generated responses average fewer than 2,048 tokens, the limited gains suggest that $K = 128$ at $B = 16$ already provides sufficient supervision across response positions. Doubling K adds more overlapping blocks and training computation, but improves acceptance length by at most 0.02 tokens on each task. We therefore use $K = 128$.

Table 5: Average acceptance length τ versus the number of sampled training anchors; D3D-1.7B, Qwen3-8B target, $B = 16$.

K	MATH-500	HumanEval	MT-Bench
16	7.89	6.00	4.41
64	8.14	6.21	4.45
128	8.15	6.24	4.50
256	8.17	6.24	4.51

C Selective Prefilling with D3D

Our main results use a trained standalone diffusion drafter for full-prompt asynchronous speculative decoding. For each drafter size, we use a single trained checkpoint for both prompt selection and sequential speculative decoding. Prefilling fewer tokens reduces target computation and can further improve end-to-end latency. Because the target conditions on a reduced prompt, we report task scores alongside latency speedups to quantify the quality–latency trade-off. Decoding remains target-verified for the retained prompt. SpecPrefill (Liu et al., 2025) suggests reusing its token selector for speculative decoding but provides no experimental results for the combination. To our knowledge, this is the first long-context evaluation in which a standalone diffusion drafter performs both prompt selection and speculative drafting.

Inference setting. Figure 5 illustrates the inference workflow. For a prompt $x = (x_1, \ldots, x_L)$, we run the standalone drafter on the full prompt and score prompt chunks with a one-MASK importance probe. Given a target retention fraction $\rho \in \{0.1, 0.2, 0.3\}$, we select the highest-scoring chunks to retain approximately ρL tokens, preserving their original order to obtain x_ρ. The target model prefills x_ρ; the same drafter then prepares blocks of size 16 (one target anchor and 15 drafted tokens) for sequential, target-verified speculative decoding. The drafter maintains its own prefix cache, separate from the target cache built on x_ρ. Greedy verification follows the target model conditioned on x_ρ; selecting x_ρ can change the output relative to conditioning on the original prompt x.

Evaluation. We evaluate Qwen3-32B on 18 datasets across five task categories from the original LongBench (Bai et al., 2024). We compare full-prompt sequential D3D drafting with selective-prefill variants retaining 10%, 20%, or 30% of the prompt. Tables 6 and 7 report the LongBench score for each category and the speedup in time to first token (TTFT), decoding, and end-to-end (E2E) latency relative to full-prompt AR decoding in the same category. "Full" denotes the complete

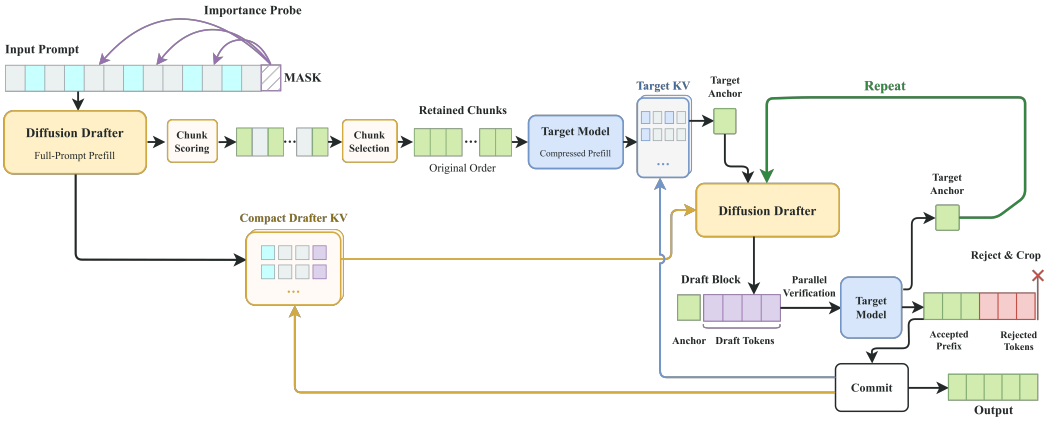

Figure 5: Selective prefilling and sequential speculative decoding with D3D. The same standalone drafter scores prompt chunks for target prefilling and generates candidate blocks for verification. Retained chunks preserve their original order, and the drafter and target maintain separate KV caches.

prompt; percentages denote the target retention fraction. Higher is better in every column. Category scores use the corresponding LongBench task metrics and should be compared within a category.

Table 6: LongBench results with a 1.7B D3D drafter and Qwen3-32B target model.

Task	Setting	Score	TTFT	Decode	E2E
Single-doc QA	Full + D3D	47.91	0.84×	3.09×	1.46×
	30% + D3D	44.23	2.09×	3.81×	2.83×
	20% + D3D	42.46	2.57×	3.95×	3.22×
	10% + D3D	41.27	2.91×	4.25×	3.56×
Multi-doc QA	Full + D3D	45.98	0.86×	2.24×	1.51×
	30% + D3D	44.32	2.01×	2.07×	2.05×
	20% + D3D	41.27	2.71×	2.47×	2.54×
	10% + D3D	40.68	2.84×	2.28×	2.42×
Summarization	Full + D3D	24.07	0.81×	1.82×	1.76×
	30% + D3D	22.71	1.74×	2.24×	2.23×
	20% + D3D	21.88	2.55×	2.62×	2.62×
	10% + D3D	21.17	3.54×	2.98×	3.00×
Few-shot learning	Full + D3D	57.07	0.85×	2.26×	1.79×
	30% + D3D	57.72	1.30×	1.64×	1.57×
	20% + D3D	57.79	2.41×	2.38×	2.39×
	10% + D3D	55.84	3.25×	2.43×	2.53×
Code completion	Full + D3D	14.73	0.90×	1.69×	1.52×
	30% + D3D	13.62	2.05×	2.15×	2.14×
	20% + D3D	13.19	2.43×	2.14×	2.16×
	10% + D3D	13.28	2.71×	2.01×	2.07×

Table 7: LongBench results with an 8B D3D drafter and Qwen3-32B target model.

Task	Setting	Score	TTFT	Decode	E2E
Single-doc QA	Full + D3D	47.91	0.72×	4.16×	1.40×
	30% + D3D	47.42	1.50×	5.49×	2.62×
	20% + D3D	46.41	1.70×	5.96×	2.93×
	10% + D3D	47.05	1.93×	6.73×	3.32×
Multi-doc QA	Full + D3D	45.97	0.74×	2.82×	1.54×
	30% + D3D	46.25	1.54×	3.15×	2.40×
	20% + D3D	46.95	1.80×	3.18×	2.59×
	10% + D3D	45.75	2.10×	3.24×	2.79×
Summarization	Full + D3D	24.06	0.72×	2.29×	2.16×
	30% + D3D	23.06	1.59×	3.48×	3.37×
	20% + D3D	22.16	1.78×	3.79×	3.68×
	10% + D3D	21.69	2.14×	4.29×	4.18×
Few-shot learning	Full + D3D	57.06	0.73×	2.99×	2.04×
	30% + D3D	56.19	1.60×	3.61×	3.03×
	20% + D3D	55.86	1.87×	3.72×	3.23×
	10% + D3D	55.43	2.15×	3.46×	3.16×
Code completion	Full + D3D	14.73	0.68×	2.21×	1.76×
	30% + D3D	14.16	1.54×	2.77×	2.54×
	20% + D3D	13.38	1.79×	2.84×	2.66×
	10% + D3D	13.24	1.94×	2.74×	2.62×

With Full + D3D, the target still prefills the entire prompt, while sequential execution adds drafter prefilling, first-block drafting, and verification before the first token. TTFT speedups therefore fall below $1\times$ despite faster decoding.

Selective prefilling adds substantial end-to-end gains to D3D's decoding acceleration. With the 8B drafter, 30% retention improves E2E speedup over Full + D3D in all five LongBench categories, spanning document QA, summarization, few-shot learning, and code completion, while keeping scores close to full-prompt results. The gains persist at lower retention: on single-document QA, 10% raises E2E speedup from $1.40\times$ to $3.32\times$ with a score of 47.05 versus 47.91 for Full + D3D; at 20%, multi-document QA scores 46.95 at $2.59\times$ E2E speedup. The same trained standalone drafter thus accelerates target prefilling and decoding across tasks and retention levels.

Drafter size. A larger drafter improves QA under aggressive selective prefilling. At 10% retention, the 8B drafter scores 47.05 on single-document QA versus 41.27 for the 1.7B drafter; both score 47.91 with the full prompt. The gap appears when the drafter selects the target's context, indicating that larger capacity helps preserve the information needed for accurate answers with a much shorter target prompt.

Prompt retention. A smaller target prompt can add to the E2E gain from speculative drafting. On multi-document QA with the 8B drafter, retaining 20% of the prompt scores 46.95 versus 45.97 with Full + D3D, and increases E2E speedup from $1.54\times$ to $2.59\times$. Selection reduces the target's prefill work while keeping the context needed for answering; the same drafter continues to accelerate decoding.

Speedup across tasks. The balance between prefilling and decoding determines how much decoding acceleration translates into E2E speedup. With the 8B drafter at 30% retention, summarization reaches $3.37\times$ E2E speedup, close to its $3.48\times$ decoding speedup. Single-document QA reaches $2.62\times$ E2E despite $5.49\times$ faster decoding. Summarization spends more of its runtime generating the response, while prefilling accounts for a larger share in QA. Accelerating both stages yields substantial E2E gains across these different task profiles.