D-CUT: ADAPTIVE VERIFICATION DEPTH PRUNING FOR BATCHED SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding has become a leading approach for accelerating large language model (LLM) inference without compromising output quality. Recent methods further improve single-request speedup by decoupling draft length from drafting latency via parallel generation, which produces longer drafts to achieve higher mean accepted tokens (MAT). However, as request concurrency grows, these long drafts waste more compute on rejected tokens, inflating the *verification cost* and making speculative decoding suboptimal in high-concurrency scenarios. To address this challenge, we propose **D-cut**, an adaptive pruning strategy that selects draft tokens across the entire batch, concentrating verification budget on tokens most likely to be accepted. Our **D-cut** is motivated by two observations: (1) acceptance length varies significantly across concurrent requests, so **D-cut** performs *cross-request pruning* that adaptively allocates verification budget based on draft confidence; (2) verification cost largely depends on the runtime environment (GPU, parallelism, etc.), so we incorporate a *runtime cost model* into the selection, allowing **D-cut** to adapt its pruning depth to the actual deployment. Experiments across dense and MoE models show that **D-cut** improves the average speedup from $1.26\times$ to $1.65\times$ under high concurrency, restores acceleration for dense-model configurations where long-draft baselines fall below autoregressive decoding, and achieves up to $3.0\times$ speedup over autoregressive decoding on MoE models.

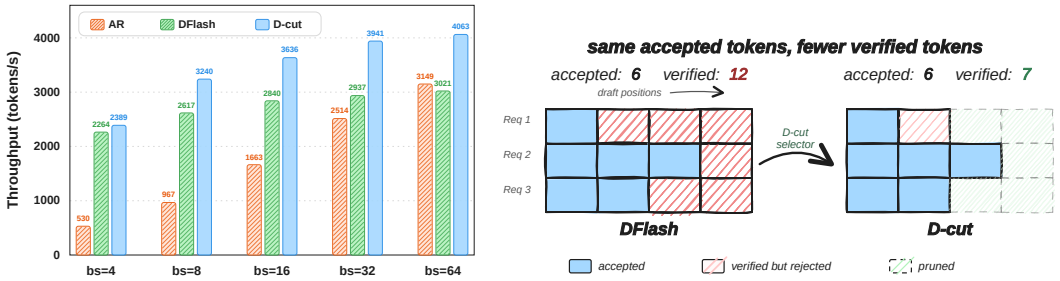
1 INTRODUCTION

Large language models (LLMs) have demonstrated strong capabilities across a wide range of tasks (OpenAI Team, 2026; DeepSeek-AI, 2026; Qwen Team, 2026). However, their autoregressive decoding (AR) paradigm, which generates one token per forward pass, limits inference throughput because each step must load the full model parameters from memory (Pope et al., 2022). This memory-bandwidth bottleneck has motivated extensive research into accelerating LLM inference.

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) has emerged as a leading lossless acceleration approach. The core idea is *draft-then-verify*: a lightweight drafter proposes multiple candidate tokens, which the target model verifies in a single forward pass. In low-concurrency settings, verification can process many tokens in parallel at roughly the same cost as generating one, so each accepted draft token effectively saves a full decoding step. The speedup is governed by two factors: the *acceptance length* (how many draft tokens are accepted per step) and the *step time* (how expensive verification is per step) (Xia et al., 2024).

Recent research has sought to improve speculative decoding by increasing the draft length and acceptance rate while minimizing drafting latency. Early methods rely on independent small models as drafters (Leviathan et al., 2023; Chen et al., 2023; Liu et al., 2025; Zhang et al., 2024a). To improve draft accuracy without introducing excessive overhead, subsequent work proposes reusing the target model's hidden features to drive lightweight, single-layer autoregressive drafters (Li et al., 2025a; Du et al., 2024; Cai et al., 2024; Gloeckle et al., 2024). More recently, this feature-reuse approach has evolved into block-parallel drafters that generate an entire block of draft tokens in a single forward pass, effectively decoupling draft length from drafting latency (Chen et al., 2026; Hui et al., 2026; Huang et al., 2026; Zhang et al., 2026). With parallel drafting, these block-wise drafters

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.



(a) Throughput under increasing batch size.

(b) Cross-request verification-depth pruning.

Figure 1: **Motivation and overview of D-cut.** (a) Throughput on Math500 with Qwen3-8B on a single H20 GPU. DFlash benefits from long block-parallel drafts at small batch sizes, but saturates as batched verification becomes compute-bound and falls below AR decoding at bs=64. (b) **D-cut** treats all draft tokens in a batch as sharing one verification budget, ranking drafts by confidence and pruning low-confidence suffixes instead of verifying the same depth for every request.

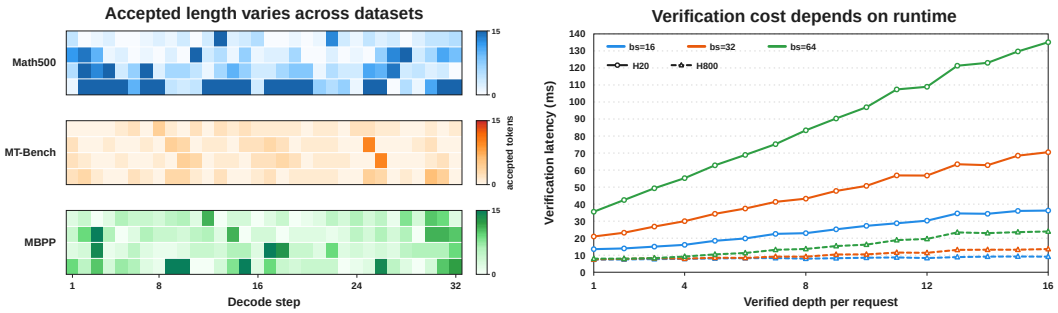
generate longer drafts and achieve higher mean accepted tokens (MAT), leading to better speedups at low concurrency.

However, **as request concurrency grows, the speedup from speculative decoding shrinks and can even fall below standard autoregressive decoding** (Liu et al., 2026d). In batched serving, the target model must verify $B \times (D+1)$ tokens per step, where B is the batch size and D is the draft length. As the batch size increases, verification transitions from memory-bound to compute-bound, and the cost of processing each additional draft token becomes non-negligible (Hu et al., 2026). This cost growth is compounded by low draft utilization: for DFlash with $D = 15$, the average acceptance length is only $\sim$6 tokens, meaning roughly 60% of verified draft tokens are rejected. Figure 1a quantifies this effect: DFlash delivers large speedups at small batch sizes, but as concurrency increases, its throughput saturates while AR keeps scaling, until at bs $= 64$ it falls below standard autoregressive decoding.

To address this challenge, we propose **D-cut**, an adaptive pruning strategy that selectively retains draft tokens across the entire batch, concentrating verification compute on positions most likely to be accepted. As illustrated in Figure 1b, **D-cut** treats all draft tokens in a batch as sharing one verification budget and prunes low-confidence drafts per request. It is built on two mechanisms: First, **cross-request pruning** relaxes the uniform-depth constraint. Given the significant variation in acceptance lengths across a batch, **D-cut** leverages the drafter's prediction confidence to globally rank all candidate tokens, shifting verification budget from low-confidence suffixes to high-confidence continuations across requests. Second, **runtime-adaptive depth selection** aligns the pruning strategy with actual runtime conditions. Recognizing that the latency penalty of verifying extra tokens depends heavily on the runtime environment, including model architecture, parallelism strategy, and GPU type, **D-cut** profiles this cost at startup. It then incorporates the runtime-specific cost into its selection objective to choose a suitable pruning depth for the current deployment.

Our main contributions are as follows:

- We identify the *verification cost explosion* problem in batched speculative decoding, showing that DFlash can fall below the autoregressive baseline at high concurrency, a mismatch between cheap long-draft generation and costly batched verification of rejected tokens.
- We propose **D-cut**, which performs cross-request pruning guided by draft confidence and runtime-adaptive depth selection via a profiled cost model. **D-cut** is training-free, needs no change to the target model, and preserves the target's output distribution.
- We evaluate **D-cut** on dense and MoE models across batch sizes, showing that it recovers acceleration for dense configurations where DFlash falls below the autoregressive baseline, and reaches up to $3.0\times$ speedup on MoE models.



(a) Accepted-length variation across datasets.

(b) Runtime verification cost profiling.

Figure 2: **Motivating observations for D-cut.** (a) MAT trajectories for four representative requests in DFlash (Qwen3-8B, $D{=}15$). Accepted length varies substantially across requests, steps, and datasets, motivating *cross-request allocation* of the verification budget. (b) P50 verification latency versus verified depth per request on H20 and H800. The cost curves change with batch size, GPU type, model architecture, and parallelism, motivating *runtime-adaptive depth selection*.

2 Motivation

2.1 Accepted Length Varies Across Requests

Recent studies on autoregressive drafters have established that the number of accepted draft tokens varies significantly across requests and decoding steps (Liu et al., 2025; 2026b; Shen et al., 2026b). A natural question arises: *does this high variance persist for block-parallel diffusion drafters like DFlash?* To investigate this, we tracked the accepted length of four representative requests during a stable high-concurrency window, using DFlash with draft length $D{=}15$ on Qwen3-8B.

Figure 2a shows that block-parallel drafters exhibit the same high variance. Within a single decoding step, one request may accept more than 12 draft tokens while another accepts fewer than 3, yet both pay the same verification cost of $D{+}1{=}16$ positions. The same request swings between long and short acceptance across steps, and the overall pattern varies by benchmark: Math500 (Lightman et al., 2024) and MBPP (Austin et al., 2021) produce frequent long-acceptance sequences, while MT-Bench (Zheng et al., 2023) is dominated by short prefixes where most full-depth verification is wasted. Moreover, similar to autoregressive drafters (Li et al., 2024; Wang et al., 2024), the block-diffusion drafter's own prediction confidence provides a lightweight proxy for whether a draft token will be accepted by the target model. (Validated in Appendix D.)

Despite this variance, block-parallel methods such as DFlash and P-EAGLE (Hui et al., 2026) verify a fixed draft depth for every request in the batch. Such a fixed depth must be set conservatively to accommodate long-acceptance requests, wasting target-model compute on the low-yield suffixes of others; verification is better allocated at the granularity of individual draft tokens. Rather than a uniform depth, **D-cut** uses the drafter's confidence to globally rank candidate tokens across the batch, retaining high-confidence drafts while pruning low-confidence suffixes.

2.2 Verification Cost Depends on Runtime

Having established *which* tokens to verify, the second question is *how much* to prune. A natural intuition is that the right amount of pruning is tied to the runtime environment, because the cost of verifying extra draft tokens differs sharply across deployments. Figure 2b profiles P50 verification latency while varying the verified depth per request from 1 to 16. On the H20, a compute-limited GPU, latency rises steeply with depth: at batch size 16, raising the depth from 4 to 16 adds about 20 ms per step (from 16 to 36 ms), so pruning should be aggressive to avoid inflating step time. On a compute-rich GPU such as the H800, the same range adds barely 1 ms (from 8 to 9 ms) and the curve stays nearly flat, so verifying extra tokens is cheap and pruning should be conservative. Model architecture (dense vs. MoE), parallelism strategy (TP/DP), and batch size shift these cost curves in the same way (Chen et al., 2025; Sadhukhan et al., 2025). A single fixed pruning strategy can therefore work well in one environment yet perform poorly in another.

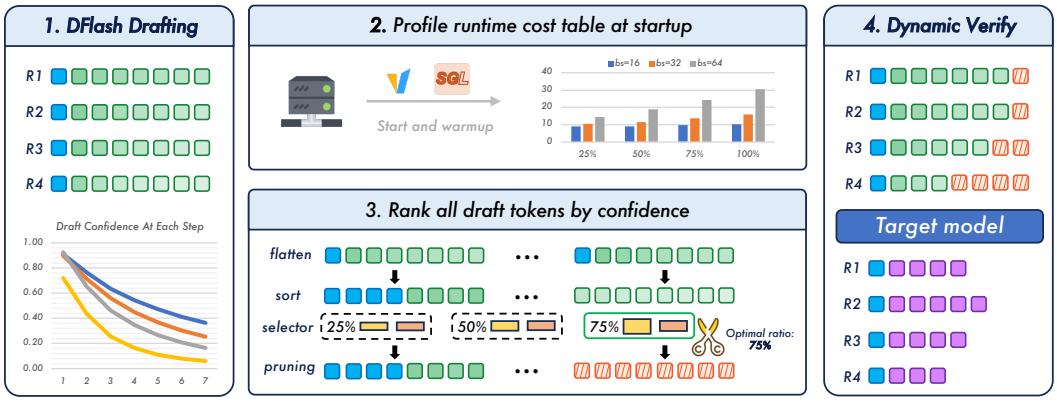


Figure 3: **Overview of D-cut.** The drafter produces batched draft blocks with token-level confidence scores. **D-cut** evaluates the expected utility of each position, dynamically selects a runtime-aware global budget, and packs only the top candidates into a dense verification batch for the target model.

A better solution is to model this runtime cost explicitly and prune against it. The right quantity to maximize is the speedup over autoregressive (AR) decoding, which factors into a benefit term and a cost term. Writing the verification budget as K, the number of block positions retained for verification across the batch, the speedup at a given batch size is

$$\text{Speedup}(K) = \frac{\text{MAT}(K)}{T_{\text{spec}}(K)/T_{\text{AR}}}, \tag{1}$$

where $\text{MAT}(K)$ is MAT per step, $T_{\text{spec}}(K)$ is the full latency of one speculative step (drafting, selection, verification and others), and T_{AR} is the latency of one AR decoding step. **This formulation naturally decouples the speedup into two orthogonal factors:** The numerator $\text{MAT}(K)$ captures the algorithmic benefit, which depends on the drafter's accuracy and is independent of the hardware; it can be estimated dynamically using confidence scores (Figure 7). The overhead $T_{\text{spec}}(K)$ in the denominator is purely a hardware characteristic, growing with the verification budget K but independent of the drafter's acceptance rate. Because the algorithmic benefit and hardware overhead are decoupled, they can be modeled separately. **D-cut** estimates the benefit via draft confidence at runtime and profiles the hardware cost curve once at startup, combining them to dynamically find the optimal budget K that maximizes Equation (1).

3 Method

D-cut reduces the verification cost of block-parallel speculative decoding by framing draft pruning as a dynamic budget allocation problem. As identified in Section 2, an optimal pruning strategy must account for both the variable acceptance lengths across requests and the runtime verification cost. We describe it based on DFlash, though the principles apply to any batched block-parallel drafter.

3.1 Problem Setup and Pipeline Overview

Consider a speculative decoding step with a batch of B active requests. For each request i, the drafter generates a block of D draft tokens $z_{i,1:D}$ in a single forward pass ($D = 15$ in our implementation). These drafts are appended to the *bonus token* $z_{i,0}$ from the previous step (the token just accepted by the target model, on which the drafter conditions), forming a candidate block of $D{+}1$ positions. Because the bonus token is guaranteed to be accepted, verifying it ensures at least one token of progress per request. Rather than verifying all D drafts for every request, **D-cut** truncates the block, submitting a prefix of length $n_i{+}1 \leq D{+}1$ to the target model.

As illustrated in Figure 3, **D-cut** determines these keep depths n_i through a two-stage process. First, it scores every candidate position in the batch by estimating its expected utility, allowing high-confidence drafts to be ranked above low-confidence ones (Section 3.2). Second, it evaluates the aggregated utility of the batch against a pre-profiled hardware cost curve, dynamically selecting a global verification budget K that maximizes the projected step speedup (Section 3.3). The top-K

scoring candidates are then packed and sent to the target model, preserving the exact target distribution while minimizing wasted compute.

3.2 Estimating Verification Utility

To globally rank candidates across the batch, **D-cut** must quantify the expected benefit of verifying each position. Let $L_i \in \{0, \dots, D\}$ be the number of draft tokens the target model would accept if the full block were verified. Because acceptance is strictly sequential, keeping the first n_i draft tokens advances a total of $A_i(n_i) = 1 + \min(L_i, n_i)$ tokens (the certain bonus plus the accepted draft prefix). The expected number of tokens advanced under keep depth n_i is entirely determined by the block's survival probabilities:

$$\mathbb{E}[A_i(n_i)] = 1 + \sum_{k=1}^{n_i} \Pr(L_i \geq k). \tag{2}$$

D-cut estimates these survival probabilities using the drafter's own prediction confidence. Let $c_{i,t} \in [0, 1]$ be the drafter's confidence for the token at draft position t. Since position k is reached only if all its preceding tokens are accepted, the probability that the block survives through depth k is given by the prefix product:

$$s_{i,k} = \prod_{t=1}^{k} c_{i,t}, \quad \text{with } s_{i,0} = 1 \text{ for the certain bonus token.} \tag{3}$$

Substituting $s_{i,k}$ into Equation (2) yields an estimated advance of $\hat{A}_i(n_i) = \sum_{k=0}^{n_i} s_{i,k}$. Each term $s_{i,k}$ elegantly isolates the expected marginal token advance from verifying position k. Because $s_{i,k}$ is monotonically decreasing with depth k, any global top-K selection over these scores will naturally yield a contiguous prefix for each request.

3.3 Runtime-Adaptive Budget Allocation

With every position scored, **D-cut** must decide the global verification budget K: the total number of positions to verify across the batch. A fixed K is suboptimal across different hardware (Section 2.2), so **D-cut** dynamically selects K to maximize the overall speedup.

To avoid the memory explosion of capturing a separate CUDA graph for every possible arbitrary K, **D-cut** restricts the budget to a small set of discrete ratios $\mathcal{R} = \{0.25, 0.50, 0.75, 1.00\}$. For a given ratio $\rho \in \mathcal{R}$ and batch size B, the allowed capacity is

$$K_\rho(B) = \max(B, \lceil \rho \cdot B(D+1) \rceil), \tag{4}$$

where the lower bound B ensures every request retains at least its bonus token. At server startup, **D-cut** profiles the hardware to build a latency cost table $C(B, \rho)$, which measures the end-to-end time of a speculative step for each discrete batch shape.

During serving, **D-cut** flattens all $B(D+1)$ candidate positions and sorts them by $s_{i,k}$. For each candidate ratio ρ, it sums the scores of the top-$K_\rho(B)$ positions, representing the expected tokens advanced by the entire batch under that budget. This sum directly estimates the algorithmic benefit (the numerator) from the speedup objective in Equation (1). For the cost denominator, the speculative step latency T_{spec} is provided by the profiled table $C(B, \rho)$. While the strict speedup denominator is $T_{\text{spec}}/T_{\text{AR}}$, the autoregressive latency T_{AR} is constant for a given batch size B and independent of the chosen ratio ρ. Because it does not affect the argmax operation, T_{AR} is omitted. **D-cut** therefore selects the optimal ratio $\rho^\star$ by maximizing:

$$\rho^\star = \arg\max_{\rho \in \mathcal{R}} \frac{\sum_{(i,k) \in \mathcal{S}_{K_\rho(B)}} s_{i,k}}{C(B, \rho)}, \tag{5}$$

where $\mathcal{S}_{K_\rho(B)}$ is the selected top-$K_\rho(B)$ set. By dynamically choosing $\rho^\star$, **D-cut** ensures that high-confidence drafts are aggressively verified when compute is cheap (e.g., on an H800), but heavily pruned when verification is expensive (e.g., on an H20) or when draft quality is poor. Crucially, because **D-cut** only drops low-utility suffixes and leaves the target model's accept/reject logic unaltered, the verified output exactly matches the target model's distribution (Appendix A).

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models and benchmarks. We evaluate six target models that cover both dense and MoE architectures: Llama-3.1-8B (Team, 2024), Qwen3-4B, Qwen3-8B, Qwen3.5-27B, Qwen3.5-35B-A3B (Qwen Team, 2026), and Hy3-295B-A21B (Tencent Hy Team, 2026). Our evaluation benchmarks span math reasoning, code generation, and multi-turn chat: GSM8K (Cobbe et al., 2021), Math500 (Lightman et al., 2024), HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and MT-Bench (Zheng et al., 2023). Unless otherwise stated, we use greedy target decoding with greedy draft proposals and target-only verification, sweep the serving concurrency over $\{4, 8, 16, 32, 64\}$, and report main results at concurrency 32; Section 4.5 changes only the target sampling temperature to 1.

Baselines and implementation. Our primary baseline is DFlash (Chen et al., 2026), which uses a block-diffusion drafter to propose an entire block of draft tokens in one forward pass. We evaluate two block sizes 8 and 16, denoted as (**8**) and (**16**) respectively. For broader context across SOTA methods, we further benchmark EAGLE-3 (Li et al., 2025b) on the 4B/8B targets, MTP (Qwen Team, 2026; Tencent Hy Team, 2026) on the Qwen3.5 and Hy3 targets using each model's default configuration, and standard autoregressive (AR) decoding as the throughput reference. All methods run on a single node of 8× NVIDIA H20 GPUs; the cross-hardware study in Section 4.4 repeats the Qwen3-8B block-16 sweep on 8× H800 GPUs. Serving details are deferred to Appendix D.

Metrics. We use two metrics in our experiments: Mean accepted tokens (MAT, denoted τ) and throughput speedup (denoted $Spd.$) over the AR baseline. We treat throughput speedup as the primary metric: under batched serving a longer draft raises MAT but also enlarges the verification batch, so a high MAT does not necessarily translate into higher throughput (Liu et al., 2026b). Reporting both separates the algorithmic benefit of pruning from the serving efficiency it actually realizes.

4.2 MAIN RESULTS

Table 1 demonstrates that long, fixed-depth verification can yield high MAT, but this does not reliably translate into throughput gains. For instance, while DFlash (16) generally accepts more tokens per step than DFlash (8), the inflated target-verification batch can cause it to run slower than standard autoregressive decoding on several benchmarks. This empirically validates our core motivation (Section 2.2): accepted length alone is an insufficient metric because the runtime verification cost dictates whether computing those extra tokens is actually worthwhile.

D-cut resolves this trade-off by selectively pruning low-utility suffixes while preserving high-confidence prefixes. Compared to DFlash (16), **D-cut** (16) accelerates throughput in 29 out of 30 model-dataset configurations, elevating the average speedup from $1.26\times$ to $1.65\times$. These gains are most pronounced where DFlash (16) achieves solid MAT but suffers from poor runtime efficiency, such as Llama-3.1-8B on GSM8K and Qwen3-8B on MT-Bench. By dynamically trimming the draft, **D-cut** retains the high acceptance benefits of long proposals while bypassing their excessive verification costs.

Even under the shorter (8) setting, **D-cut** (8) still outperforms its baseline in 25 out of 30 pairs, raising the average speedup from $1.56\times$ to $1.67\times$; the gains are smaller because an 8-token proposal leaves fewer low-utility positions to remove.

4.3 SCALING WITH CONCURRENCY

While Table 1 fixes concurrency at 32, Figure 4 illustrates absolute output-token throughput as concurrency scales from 4 to 64. Throughput naturally rises with concurrency across methods, but **D-cut** and DFlash diverge significantly at the high end. On dense targets, the long-draft DFlash (16) gains the least from batching, as the high cost of verifying long drafts effectively cancels out the benefits of accepted tokens. In contrast, **D-cut** (16) aggressively prunes low-utility suffixes, sustaining much higher throughput across the high-concurrency regime, with the performance gap widening as load increases. On the two MoE targets, **D-cut** and DFlash track closely on Qwen3.5-35B-A3B, whereas

Table 1: **End-to-end serving results at concurrency 32.** τ: mean accepted tokens (MAT). Spd.: output-token throughput speedup over autoregressive decoding. Avg.: averages over the five datasets. **Bold** marks the best speedup per column within each model. Shaded rows denote **D-cut**, green for the (8) budget and blue for the (16) budget.

Model	Method	GSM8K		Math500		HumanEval		MBPP		MT-Bench		Avg.	
		τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.
Llama-3.1 8B	EAGLE-3	2.41	0.87×	1.26	0.66×	2.47	1.03×	3.14	**1.62**×	2.23	0.97×	2.30	1.03×
	DFlash (**8**)	4.32	0.91×	4.72	1.25×	4.58	1.00×	4.75	1.30×	3.83	1.04×	4.44	1.10×
	D-cut (**8**)	4.08	**1.13**×	4.41	**1.48**×	4.30	**1.12**×	4.39	1.54×	3.49	**1.34**×	4.13	**1.32**×
	DFlash (**16**)	4.46	0.56×	4.96	0.74×	4.80	0.66×	5.06	0.77×	3.94	0.60×	4.64	0.67×
	D-cut (**16**)	3.93	0.94×	4.69	1.25×	4.33	0.98×	4.33	1.30×	3.48	1.13×	4.15	1.12×
Qwen3 4B	EAGLE-3	2.48	1.07×	2.36	1.44×	2.34	1.28×	2.26	1.42×	2.11	**1.21**×	2.31	1.28×
	DFlash (**8**)	4.85	1.16×	5.67	1.67×	4.88	1.31×	4.53	1.46×	2.84	0.84×	4.55	1.29×
	D-cut (**8**)	4.67	**1.33**×	5.34	**1.77**×	4.71	**1.44**×	4.38	**1.58**×	2.73	1.06×	4.37	**1.44**×
	DFlash (**16**)	6.14	0.90×	7.87	1.38×	6.41	0.99×	5.88	1.08×	3.23	0.56×	5.91	0.98×
	D-cut (**16**)	5.80	1.22×	6.95	1.72×	5.98	1.35×	5.48	1.47×	3.05	0.99×	5.45	1.35×
Qwen3 8B	EAGLE-3	2.31	0.96×	2.26	1.19×	2.26	1.08×	2.08	1.25×	1.95	1.08×	2.17	1.11×
	DFlash (**8**)	4.87	0.99×	5.69	1.42×	5.06	1.14×	4.61	1.26×	2.91	0.78×	4.63	1.12×
	D-cut (**8**)	4.47	**1.21**×	5.20	**1.51**×	4.51	**1.26**×	4.07	**1.50**×	2.81	**1.09**×	4.21	**1.31**×
	DFlash (**16**)	6.10	0.75×	7.76	1.10×	6.41	0.82×	5.59	0.86×	3.19	0.48×	5.81	0.80×
	D-cut (**16**)	5.68	1.08×	6.86	1.49×	5.77	1.18×	4.75	1.32×	3.01	0.97×	5.21	1.21×
Qwen3.5 27B	MTP	2.85	1.33×	2.86	1.37×	2.83	1.32×	2.73	**1.29**×	2.53	**1.22**×	2.76	**1.31**×
	DFlash (**8**)	5.61	1.40×	5.88	1.41×	5.63	1.16×	4.80	1.18×	3.42	0.88×	5.07	1.21×
	D-cut (**8**)	5.26	1.41×	5.46	1.38×	5.38	1.32×	4.27	1.09×	3.21	0.98×	4.72	1.24×
	DFlash (**16**)	7.17	1.14×	7.71	1.09×	7.63	1.12×	5.58	0.78×	3.71	0.57×	6.36	0.94×
	D-cut (**16**)	6.55	**1.44**×	6.93	**1.44**×	7.04	**1.38**×	5.06	1.11×	3.47	0.90×	5.81	1.25×
Qwen3.5 35B-A3B	MTP	2.81	1.71×	2.83	1.86×	2.79	1.77×	2.71	1.75×	2.48	1.66×	2.72	1.75×
	DFlash (**8**)	5.27	2.65×	5.41	2.75×	5.29	2.39×	4.39	2.19×	3.21	1.78×	4.71	2.35×
	D-cut (**8**)	5.02	2.45×	5.16	2.75×	5.06	2.34×	4.25	**2.24**×	3.18	**1.81**×	4.53	2.32×
	DFlash (**16**)	6.48	**2.78**×	6.93	2.95×	6.77	2.41×	5.16	2.06×	3.46	1.59×	5.76	2.36×
	D-cut (**16**)	6.20	2.66×	6.53	**3.04**×	6.60	**2.77**×	4.98	2.21×	3.36	1.80×	5.53	**2.50**×
Hy3 295B-A21B	MTP	1.86	1.36×	1.89	1.38×	1.87	1.36×	1.84	1.37×	1.68	1.25×	1.83	1.34×
	DFlash (**8**)	4.86	2.50×	4.79	2.59×	4.78	2.42×	4.56	2.47×	2.87	1.65×	4.37	2.33×
	D-cut (**8**)	4.81	2.51×	4.66	2.64×	4.71	2.50×	4.47	2.57×	2.86	**1.77**×	4.30	2.40×
	DFlash (**16**)	5.87	1.88×	5.69	2.12×	5.84	2.02×	5.33	1.94×	3.00	1.21×	5.15	1.83×
	D-cut (**16**)	5.53	**2.61**×	5.54	**2.79**×	5.63	**2.61**×	5.15	**2.64**×	2.98	1.73×	4.97	**2.48**×

Hy3-295B-A21B benefits clearly from pruning, especially under the longer (16) budget. One plausible contributor is target-side attention: Qwen3.5-35B-A3B interleaves three Gated DeltaNet layers per gated full-attention layer, while Hy3-295B-A21B uses GQA with 64 query and 8 key-value heads (Qwen Team, 2026; Tencent Hy Team, 2026). Model scale and tensor parallelism also differ, so this does not isolate the attention effect; **D-cut** therefore selects the budget from profiled runtime cost rather than architecture labels.

4.4 Budget Selection and Runtime Adaptivity

To isolate the impact of our budget-selection rule, we evaluate Qwen3-8B with the block-16 drafter across five configurations: unpruned DFlash, three fixed-ratio variants of **D-cut** (using a global batch budget of 0.25, 0.50, or 0.75 of all block positions), and the fully runtime-adaptive **D-cut** (auto). Table 2 reports the resulting throughput across concurrencies on both H20 and H800 GPUs.

The results reveal that the optimal pruning ratio is highly dynamic, varying by both benchmark and hardware. Across the contended regime in Table 2, pruning improves over the unpruned DFlash baseline in many settings, yet no single fixed ratio is best everywhere. On the compute-constrained H20, verification is a severe bottleneck: MT-Bench heavily rewards aggressive pruning (**D-cut** (0.25) reaches $2{,}564$ tokens/s at concurrency 64), whereas GSM8K and HumanEval favor a more moderate 0.50 ratio. On the compute-rich H800, verifying extra tokens is much cheaper, so the larger 0.50 budget becomes optimal for GSM8K and HumanEval and the 0.25 ratio turns wasteful, while

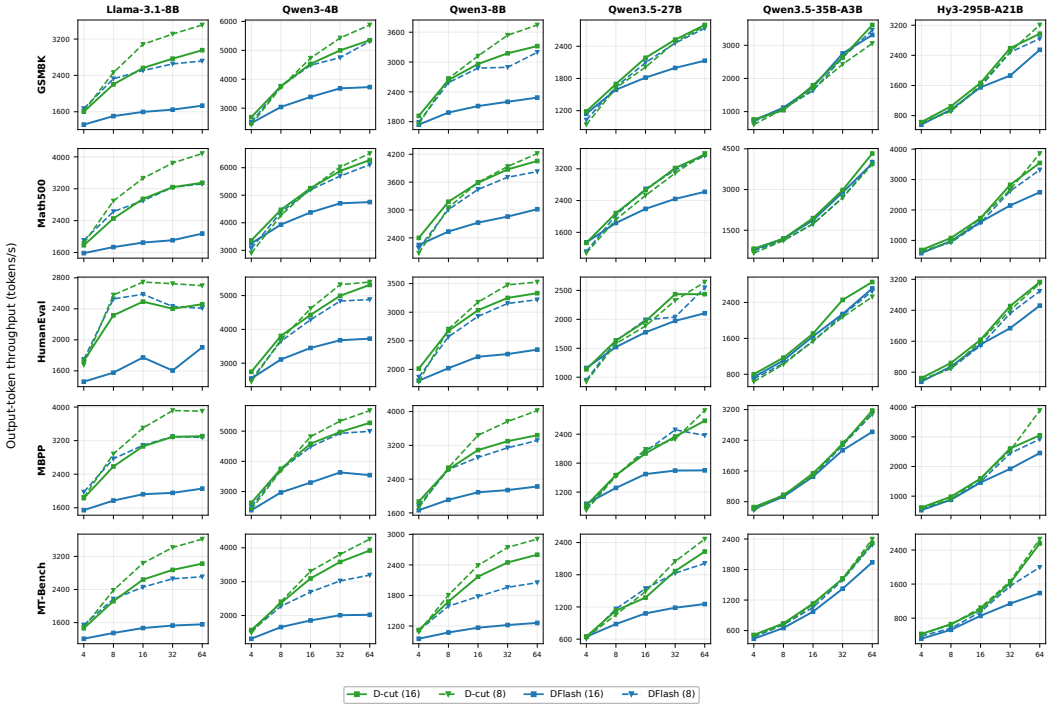


Figure 4: **Absolute output-token throughput (tokens/s) as concurrency grows from 4 to 64.** Rows are datasets, columns are target models. Green is **D-cut**, blue is DFlash; solid lines are the (16) budget, dashed the (8). On the dense targets, the long-draft DFlash (16) gains the least at high concurrency, whereas **D-cut** sustains higher throughput. The MoE targets differ: the two track closely on Qwen3.5-35B-A3B, while Hy3-295B-A21B benefits more clearly from pruning.

Table 2: **Throughput ablation on the budget-selection rule (Qwen3-8B).** Output-token throughput (tokens/s) per benchmark at concurrencies 16–64, on H20 and H800. **D-cut** (0.25/0.5/0.75) are fixed-ratio variants that verify a global batch budget equal to that fraction of all $B(D{+}1)$ block positions; **D-cut** (auto) selects the budget from the profiled runtime cost. The best budget per column is shaded blue (**bold**) and the runner-up green (underlined). On the weaker H20, verification is the bottleneck and aggressive pruning wins; on the compute-rich H800, the best fixed ratio shifts toward retaining more positions, while **D-cut** (auto) stays at or near the top on both devices.

		GSM8K			HumanEval			MT-Bench		
Device	**Method**	bs=16	bs=32	bs=64	bs=16	bs=32	bs=64	bs=16	bs=32	bs=64
H20	DFlash	2115	2200	2285	2221	2268	2346	1167	1220	1261
	D-cut (0.25)	2764	3014	3123	2754	2972	3127	2152	2377	2564
	D-cut (0.5)	2891	3114	3147	3005	3179	**3351**	1729	1856	1941
	D-cut (0.75)	2427	2618	2630	2463	2716	2796	1355	1442	1514
	D-cut (auto)	**2961**	**3175**	**3320**	**3035**	**3250**	3333	**2169**	**2449**	**2600**
H800	DFlash	5963	7912	7908	6492	8167	8076	3159	4147	4245
	D-cut (0.25)	4814	5479	6143	4735	7251	9520	3387	**5495**	**7126**
	D-cut (0.5)	**6615**	**9780**	10824	6799	9527	10831	**3562**	5185	6041
	D-cut (0.75)	6541	8661	9551	6809	8962	9827	3404	4594	5049
	D-cut (auto)	6391	9663	**11194**	**6823**	**9737**	**11347**	3469	5253	6894

MT-Bench still benefits from aggressive pruning at the high-concurrency end because its accepted prefixes are shorter.

D-cut (auto) tracks this shifting optimum without requiring manual, per-deployment tuning. By reading the profiled cost table at startup and dynamically selecting the budget, it stays near the best fixed ratio across devices and benchmarks. For instance, at concurrency 64 on H800 (GSM8K),

D-cut (auto) achieves 11,194 tokens/s, outperforming both the best fixed ratio (10,824) and DFlash (7,908). Averaged across the three benchmarks and concurrencies 16 to 64, **D-cut** (auto) accelerates DFlash by $1.58\times$ on the H20 and $1.25\times$ on the H800, matching the best fixed configuration at the aggregate level without requiring the ratio to be chosen in advance.

4.5 Robustness across Sampling Regimes

The results so far use greedy target decoding. Figure 6 repeats the Qwen3-8B H20 sweep with temperature-1 target sampling, retaining greedy draft proposals and target-only verification. Stochastic target sampling reduces draft utilization, which severely penalizes long, fixed-depth proposals. Averaged over the three benchmarks and concurrencies 16 to 64, DFlash's speedup collapses to $0.68\times$, falling below the autoregressive baseline at high concurrency. By pruning the now less useful suffixes, **D-cut** (auto) sustains $1.15\times$ over that baseline, a $1.68\times$ relative gain over DFlash. This exceeds the $1.58\times$ gain under greedy decoding (Section 4.4), so harsher sampling regimes make adaptive pruning more critical, not less.

5 Related Work

Speculative decoding accelerates autoregressive generation through a lossless *draft-then-verify* paradigm (Leviathan et al., 2023; Chen et al., 2023), and a long line of work improves the drafter, from token trees (Miao et al., 2024; Chen et al., 2025; Wang et al., 2024) and multi-head prediction (Cai et al., 2024; Gloeckle et al., 2024) to feature-reusing autoregressive drafters (Li et al., 2025a; 2024; 2025b; Du et al., 2024) and block-parallel diffusion drafters such as DFlash, which emit a whole draft block in one forward pass (Chen et al., 2026). Most prior adaptive methods tune the draft length or tree online (Huang et al., 2025; Mamou et al., 2024; Liu et al., 2026b; Shen et al., 2026c; Xiong et al., 2024), but their control objective remains each request's local draft structure. Closest to our setting, recent work shows speculative speedups shrink under high concurrency as batched verification turns compute-bound (Liu et al., 2026d; Hu et al., 2026); PEARL overlaps drafting and verification with adaptive draft length (Liu et al., 2025), and ECHO casts it as budgeted tree scheduling under a global verification cap (Hu et al., 2026). **D-cut** shares the view that verification compute is the scarce resource in batched serving, but targets a different drafter class: ECHO schedules tree depth and width over tree-structured or autoregressive drafters (Gloeckle et al., 2024; Li et al., 2026), whereas DFlash emits a single linear block per request, for which the key decision is the per-request verification depth. **D-cut** therefore prunes linear draft blocks across requests and sets the depth from a profiled runtime cost model. ECHO and Graft schedule tree structure that a linear block does not have, so neither transfers to DFlash without redesign and we do not run them as baselines. Concurrent with our work, DSpark shares **D-cut**'s view of cross-request pruning and runtime-adaptive verification (Cheng et al., 2026). It uses a semi-autoregressive drafter and a trained confidence head, whereas **D-cut** reuses confidence from an existing block-parallel drafter without additional training. Graft also prunes low-confidence positions, but refills the freed budget with retrieved candidates within each request (Shen et al., 2026a), whereas **D-cut** redistributes it *across* requests. A full discussion is given in Appendix E.

6 Conclusion and Limitations

As large language models are deployed at scale, speculative decoding must shift from maximizing single-request acceptance lengths to managing batched verification efficiency. Block-parallel drafters, effective in isolation, degrade throughput at high concurrency because their long, fixed-depth drafts waste target-model compute on low-yield tokens.

D-cut treats target verification as a budgeted resource, modeling the speedup as a ratio of confidence-estimated benefit (MAT) to hardware-profiled runtime cost and pruning low-utility suffixes across the batch, so deeper drafts are attempted only when the hardware and concurrency level make it profitable. The acceleration happens entirely at the scheduling level, preserving the target model's output distribution. Across dense and MoE architectures, **D-cut** keeps long-draft configurations from falling below autoregressive baselines at high load, lifting average speedups from $1.26\times$ to $1.65\times$. The main cost is system integration: the varying verification shape needs engine support our runs did not enable, so the reported throughput is conservative (Appendix C).

Reproducibility Statement

Algorithm 1 and Appendix A give the per-step selection procedure, its complexity, and the prefix property the packing relies on. Appendix C describes the packing scheme and the startup profiling protocol, with the measured cost tables for the Qwen3-8B H20 and H800 runs. Appendix D lists the draft checkpoint and serving configuration used for every target and baseline. Appendix B states the conditions under which the selector leaves the target output distribution unchanged.

AI Use Statement

Generative AI tools were used to polish writing and to draft parts of the abstract, the conclusion, and the appendix discussion of engine compatibility. They were not used for any task in ICLR 2027's required-disclosure category: the method, the cost model, the experimental design, and the implementation are the authors' own work. We have reviewed all AI-assisted text against our implementation and measurements, and we take responsibility for the final content of this paper.

References

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

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 *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

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

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding, 2025. URL https://arxiv.org/abs/2402.12374.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John

Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro/blob/main/DeepSeek_V4.pdf.

Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, and Yang You. Glide with a cape: a low-hassle method to accelerate speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

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

Xinyi Hu, Yuhao Shen, Baolin Zhang, Hengxin Zhang, Jun Dai, Shuang Ge, Lei Chen, Yue Li, and Mingcheng Wan. Echo: Elastic speculative decoding with sparse gating for high-concurrency scenarios, 2026. URL https://arxiv.org/abs/2604.09603.

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.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. Specdec++: Boosting speculative decoding via adaptive candidate lengths, 2025. URL https://arxiv.org/abs/2405.19715.

Mude Hui, Xin Huang, Jaime Campos Salas, Yue Sun, Nathan Pemberton, Xiang Song, Ashish Khetan, and George Karypis. P-eagle: Parallel-drafting eagle with scalable training, 2026. URL https://arxiv.org/abs/2602.01469.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yucheng Li, Huiqiang Jiang, Yang Xu, Jianxin Yang, Yi Zhang, Yizhong Cao, Yuhao Shen, Fan Zhou, Rui Men, Jianwei Zhang, et al. Breaking entropy bounds: Accelerating rl training via mtp with rejection sampling. *arXiv preprint arXiv:2606.12370*, 2026.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.422. URL https://aclanthology.org/2024.emnlp-main.422/.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty, 2025a. URL https://arxiv.org/abs/2401.15077.

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

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/aca97732e30bcf1303bc22ac3924fd16-Paper-Conference.pdf.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. PEARL: Parallel speculative decoding with adaptive draft length. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=QOXrVMiHGK.

Tianyu Liu, Qitan Lv, Hao Li, Xing Gao, Xiao Sun, and Xiaoyan Sun. LogitSpec: Accelerating retrieval-based speculative decoding via next next token speculation. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 33070–33092, San Diego, California, United States, July 2026a. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1655. URL https://aclanthology.org/2026.findings-acl.1655/.

Tianyu Liu, Qitan Lv, Yuhao Shen, Xiao Sun, and Xiaoyan Sun. Talon: Confidence-aware speculative decoding with adaptive token trees, 2026b. URL https://arxiv.org/abs/2601.07353.

Tianyu Liu, Yuhao Shen, Xinyi Hu, Baolin Zhang, Hengxin Zhang, Jun Dai, Jun Zhang, Shuang Ge, Lei Chen, Yue Li, and MingCheng Wan. When hidden states drift: Can kv caches rescue long-range speculative decoding?, 2026c. URL https://arxiv.org/abs/2604.26412.

Xiaoxuan Liu, Jiaxiang Yu, Jongseok Park, Ion Stoica, and Alvin Cheung. Speculative decoding: Performance or illusion?, 2026d. URL https://arxiv.org/abs/2601.11580.

Jonathan Mamou, Oren Pereg, Daniel Korat, Moshe Berchansky, Nadav Timor, Moshe Wasserblat, and Roy Schwartz. Dynamic speculation lookahead accelerates speculative decoding of large language models, 2024. URL https://arxiv.org/abs/2405.04304.

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*, ASPLOS '24, pp. 932–949, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703867. doi: 10.1145/3620666.3651335. URL https://doi.org/10.1145/3620666.3651335.

OpenAI Team. Gpt-5.5 system card. https://openai.com/index/gpt-5-5-system-card/, 2026. System card, OpenAI.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference, 2022. URL https://arxiv.org/abs/2211.05102.

Qwen Team. Qwen3.5: Accelerating productivity with native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

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, 2025. URL https://arxiv.org/abs/2408.11049.

Yuhao Shen, Tianyu Liu, Xinyi Hu, Quan Kong, Baolin Zhang, Jun Dai, Jun Zhang, Shuang Ge, Lei Chen, Yue Li, Mingcheng Wan, and Cong Wang. Draft less, retrieve more: Hybrid tree construction for speculative decoding, 2026a. URL https://arxiv.org/abs/2605.20104.

Yuhao Shen, Tianyu Liu, Junyi Shen, Jinyang Wu, Quan Kong, Li Huan, and Cong Wang. Double: Breaking the acceleration limit via double retrieval speculative parallelism, 2026b. URL https://arxiv.org/abs/2601.05524.

Yuhao Shen, Junyi Shen, Quan Kong, Tianyu Liu, Yao Lu, and Cong Wang. Specbranch: Speculative decoding via hybrid drafting and rollback-aware branch parallelism, 2026c. URL https://arxiv.org/abs/2506.01979.

Llama Team. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Tencent Hy Team. Hy3, 2026. URL https://huggingface.co/tencent/Hy3. Model card.

Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. Opt-tree: Speculative decoding with adaptive draft tree structure, 2024. URL https://arxiv.org/abs/2406.17276.

Zhaoxuan Wu, Zijian Zhou, Arun Verma, Alok Prakash, Daniela Rus, and Bryan Kian Hsiang Low. Tetris: Optimal draft token selection for batch speculative decoding, 2025. URL https://arxiv.org/abs/2502.15197.

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

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

Jiebin Zhang, Zhenghan Yu, Song Liu, Eugene J Yu, Zheng Li, Dawei Zhu, Jiangshan Duo, Weimin Xiong, Yifan Song, Guanghua Yu, et al. Dflare: Scaling up draft capacity for block diffusion speculative decoding. *arXiv preprint arXiv:2606.02091*, 2026.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft& verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282. Association for Computational Linguistics, 2024a. doi: 10.18653/v1/2024.acl-long.607. URL http://dx.doi.org/10.18653/v1/2024.acl-long.607.

Situo Zhang, Hankun Wang, Da Ma, Zichen Zhu, Lu Chen, Kunyao Lan, and Kai Yu. Adaeagle: Optimizing speculative decoding via explicit modeling of adaptive draft structures, 2024b. URL https://arxiv.org/abs/2412.18910.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=uccHPGDlao.

A Algorithm and Analysis

Algorithm 1 summarizes how **D-cut** executes inside a single DFlash decoding step: the selector runs after the drafter samples a block and before the target model verifies it, ranking block positions across requests, choosing a runtime-aware budget, and packing the retained prefixes for verification.

Algorithm 1 D-cut at one decoding step.

Require: Active requests $\mathcal{B}$, draft depth D, ratio buckets $\mathcal{R}$, runtime cost table C
Ensure: Packed target-verification batch
1: Run the DFlash drafter to obtain draft tokens $z_{i,1:D}$ and confidences $c_{i,1:D}$ for each request $i \in \mathcal{B}$
2: Set $s_{i,0} \leftarrow 1$ and compute prefix scores $s_{i,k} \leftarrow \prod_{t=1}^{k} c_{i,t}$ for all block positions (i,k), $k = 0, \ldots, D$
3: **for all** $\rho \in \mathcal{R}$ **do**
4: Compute $K_\rho(B)$ using Equation (4)
5: Let $\mathcal{S}_{K_\rho(B)}$ be the top-$K_\rho(B)$ positions ranked by $s_{i,k}$ (ties toward smaller k)
6: Score ρ by the utility-cost objective in Equation (5)
7: **end for**
8: Select $\rho^\star$ with the largest objective value
9: **for all** request $i \in \mathcal{B}$ **do**
10: $n_i \leftarrow \max(\{k : (i,k) \in \mathcal{S}_{K_{\rho^\star}(B)}\} \cup \{0\})$
11: Keep the block prefix $z_{i,0:n_i}$ and prune the remaining draft suffix
12: **end for**
13: Pack all retained block prefixes for target verification

Selector complexity. Each step the selector accumulates log-confidences to obtain the prefix scores of all $B(D{+}1)$ block positions (Equation (3)), which is $O(B(D{+}1))$. It then flattens the positions and extracts the top-$K_\rho(B)$ set; ranking the $B(D{+}1)$ scores costs $O\big(B(D{+}1)\log B(D{+}1)\big)$, and scoring the objective in Equation (5) for each of the $|\mathcal{R}|$ ratios is an $O(B(D{+}1))$ prefix-sum sweep over the ranked list. The per-step cost is thus $O\big(B(D{+}1)\log B(D{+}1)\big)$ and independent of the target model size, whereas verification cost grows with both the model size and the retained budget K. This asymmetry is why the measured selector cost stays a small fraction of the step (Figure 5).

Retained positions form a contiguous prefix. The selector ranks positions by the prefix-product score $s_{i,k} = \prod_{t=1}^{k} c_{i,t}$ (Equation (3)), which is non-increasing in k because $s_{i,k} = s_{i,k-1}\, c_{i,k}$ with $c_{i,k} \in [0,1]$. For a fixed request i this gives $s_{i,j} \geq s_{i,k}$ whenever $j < k$. Suppose the global top-K set $\mathcal{S}_K$ contains a position (i,k); then every shallower position (i,j) with $j < k$ has a score at least as large and is therefore also in $\mathcal{S}_K$. Hence each request's retained positions are exactly $\{0,1,\ldots,n_i\}$ with $n_i = \max\{k : (i,k) \in \mathcal{S}_K\}$, a contiguous prefix of the draft block rather than a scattered subset. This is the property that lets **D-cut** hand the verifier a valid shortened block and reuse the engine's piecewise CUDA-graph shapes: cross-request pruning only ever removes a suffix from a request, never an interior position.

Correctness. In the evaluated setup, draft proposals are deterministic, and the temperature-1 study changes only target-side sampling and uses target-only verification. **D-cut** selects the keep depths before target samples are drawn and leaves the target conditional distributions unchanged, so truncating each block to a prefix changes only how many target positions are evaluated in parallel, not the distribution of committed tokens. Progress never stalls either, since the budget is clamped to $K_\rho(B) \geq B$ (Equation (4)) so every request keeps at least its bonus token $z_{i,0}$ and advances by at least one token per step.

B Causality under Rejection Sampling

Selection bias from inclusive confidence. We now consider an extension in which each request exposes the per-position conditional proposal distributions $q_{i,k}(\cdot \mid z_{i,<k})$ required by standard rejection sampling (RS), and the target distribution p verifies proposals with independent sampling randomness across requests (Leviathan et al., 2023; Chen et al., 2023). In this setting, directly applying the original score $s_{i,k} = \prod_{t=1}^{k} q_{i,t}(z_{i,t})$ can change the output distribution: whether position

k is retained depends on the sampled proposal $z_{i,k}$ itself, violating the non-anticipating condition required by RS (Cheng et al., 2026). For a one-position example, let the vocabulary be $\{A, B\}$, with $p(A) = p(B) = 0.5$ and $q(A) = 0.9$, $q(B) = 0.1$, and retain the proposal only when $q(z) > 0.5$. When A is proposed, RS accepts it with probability $0.5/0.9 = 5/9$ and otherwise emits B from the residual distribution; when B is proposed, the selector skips the proposal and samples directly from p. The resulting probability of emitting A is therefore

$$0.9 \cdot \frac{5}{9} + 0.1 \cdot 0.5 = 0.55 \neq p(A), \tag{6}$$

showing that leaving the RS accept/reject rule unchanged is insufficient when the retention decision depends on the current proposal.

Shifted confidence with causal budget selection. To make the scheduling decision causal, we shift each draft-position score by one position:

$$\tilde{s}_{i,0} = 1, \qquad \tilde{s}_{i,k} = \prod_{t=1}^{k-1} q_{i,t}(z_{i,t}), \quad k = 1, \ldots, D. \tag{7}$$

The admission score for position k then depends only on proposals before k. The shift alone is not sufficient if the selector retrospectively chooses the budget from all scores in the current block: $z_{i,k}$ affects $\tilde{s}_{i,k+1}$ and can thereby change the selected budget, which may in turn change whether position k is retained. Our preferred solution is a causal early-stopping scan, following the non-anticipating construction in DSpark (Cheng et al., 2026). The selector maintains one frontier position per request and repeatedly considers the frontier with the largest shifted score. It evaluates the utility-cost objective at the resulting verification size; if the objective improves, the position is irrevocably admitted and the next position from that request is exposed, otherwise the scan stops. This determines K from the current block without consulting a descendant score before deciding on its ancestor. Unlike a retrospective global argmax, early stopping may miss a later optimum when the utility-cost curve is not unimodal. Unimodality affects budget optimality only, not the distributional correctness established below. When token-level stopping does not fit an asynchronous or bucketed execution path, an alternative is to estimate K from confidence statistics of previous decoding steps, following the history-based scheduling used in DSpark implementations (Cheng et al., 2026). The resulting history-determined K is fixed before the current block is observed, after which the selector applies global top-K allocation to the shifted scores. Both variants resolve ties deterministically by shallower depth and then request index, independently of proposal values. The verifier still uses the original p and q distributions in the RS correction; shifted confidence affects scheduling only.

Distributional correctness. Condition on the committed history and on the independent proposal randomness of other requests. For position (i, k), the shifted score depends only on $z_{i,<k}$, while all deeper scores from the same request are no larger than $\tilde{s}_{i,k}$. Under causal early stopping, the decision to admit (i, k) is made before any descendant score from request i is exposed; under history-based scheduling, K is fixed before the current proposals are drawn and every descendant is ranked after (i, k). In either case, changing $z_{i,k}$ or any later proposal cannot change whether (i, k) is retained. Thus the retention event is independent of $z_{i,k}$ conditional on $z_{i,<k}$, and the proposal at position k still follows $q_{i,k}$ after conditioning on retention. Standard RS then accepts with probability $\min(1, p_{i,k}(z_{i,k})/q_{i,k}(z_{i,k}))$ and, upon rejection, samples from the residual distribution proportional to $(p_{i,k} - q_{i,k})_+$; if the retained block is fully accepted, the bonus token is sampled directly from the next target distribution. The token committed at the first unresolved position therefore follows $p_{i,k}$. The shared top-K decision can couple how far different requests advance, but it does not change their conditional token distributions; with independent sampling randomness across requests, induction over positions and decoding steps recovers the target joint output distribution.

C System Implementation

Packing and CUDA-graph compatibility. The retained prefixes have unequal lengths n_i+1 across requests. **D-cut** concatenates them into a single contiguous verification batch of $K_{\rho^\star}(B)$ positions, together with the block and position metadata the target attention needs to score each retained position against its accepted prefix. Because the budget is restricted to the ratio buckets

$\mathcal{R}$ (Equation (4)), the packed batch takes one of only a few total lengths, each of which overlaps the piecewise CUDA graphs the engine already captures for ordinary decoding. Pruning therefore changes how many positions are packed, not the set of captured shapes, so it adds no extra graphs and no additional graph memory. A freely varying K, by contrast, would produce a new shape almost every step and force the engine to capture and store many more graphs.

Full-graph capture and cpu-gpu overlap. Piecewise graphs are only part of the picture. Two further mechanisms in current engines assume a static per-step shape: full-graph capture, which records the whole step as a single graph instead of a sequence of piecewise segments, and the cpu-gpu overlap path (Spec-V2), which prepares the next step's metadata while the current step is still running. Both admit known solutions, and both fit how **D-cut** is constructed. For full-graph capture, it suffices to key the graph on the total token count of the packed batch and lift the attention-metadata preparation kernel out of the graph, because Equation (4) already restricts that total to the few values $K_\rho(B)$ can take; the metadata is the only part that genuinely varies from step to step. For the overlap path, the budget can be predicted from the confidence statistics of an earlier step rather than the current one, which fixes the verification shape before the current block is drafted and leaves the overlap free to prepare the matching metadata. This is the same history-based scheduling that Appendix B uses to keep the selector non-anticipating under rejection sampling, so one mechanism buys both distributional correctness and pipeline overlap. Our measurements enable neither path: the runs in Section 4 use piecewise capture without cpu-gpu overlap. The reported throughput is therefore a conservative estimate of what **D-cut** delivers on an engine that supports both.

Cost-table profiling protocol. **D-cut** builds the cost table $C(B, \rho)$ once at server startup, before the server accepts traffic. For each batch size B that the engine captures a piecewise CUDA graph for, and each ratio $\rho \in \mathcal{R}$, we run dummy speculative steps that exercise the full pipeline, drafting, selection, packing, and target verification of the $K_\rho(B)$ retained positions, under the same model weights, parallelism, compilation mode, and CUDA-graph configuration as serving, and record the median per-step latency over a small number of repetitions. Each entry is therefore an end-to-end T_{spec} measurement rather than a verification-only cost, consistent with Equation (5). At decode time the selector reads $C(B, \rho)$ at the current batch size B in $O(1)$; batch sizes that fall between captured points reuse the nearest captured entry. Because profiling touches only dummy tensors and runs before serving, it adds a one-time startup cost and no per-request latency. Table 3 shows concrete tables from the Qwen3-8B H20 and H800 runs. The cost is nearly flat at $B = 1$, where verification is underutilized, but grows sharply with the retained ratio at larger batches, especially on the compute-constrained H20. At $B = 64$, verifying the full DFlash block costs 133.91 ms on H20 and 30.39 ms on H800, while the 25% bucket costs 55.16 ms and 14.46 ms, respectively. This changing slope across both batch size and hardware is the reason **D-cut** reads the profiled table at runtime instead of treating each extra draft position as having a constant cost.

Table 3: **Example startup cost table $C(B, \rho)$ for Qwen3-8B on H20 and H800.** Each entry is the profiled end-to-end speculative-step latency in milliseconds for a retained-position budget $K_\rho(B) = \rho B(D + 1)$, with $D = 15$ and ratio buckets $\mathcal{R} = \{25\%, 50\%, 75\%, 100\%\}$.

B	H20				H800			
	25%	50%	75%	100%	25%	50%	75%	100%
1	7.180	7.157	7.310	7.353	7.995	8.008	8.081	8.101
8	10.364	12.775	16.454	19.663	8.363	8.457	8.657	8.628
16	15.991	22.785	30.003	35.869	9.008	9.079	9.700	10.208
32	29.903	43.000	56.611	70.066	10.408	11.554	13.793	15.926
64	55.162	82.287	107.874	133.914	14.464	18.904	24.317	30.388

Selector overhead and where the time goes. **D-cut** adds one routine on top of DFlash: a global top-K ranking of the per-request draft confidences each step, after a one-time cost-table profiling at startup. Figure 5 splits the per-step latency at concurrency 64, in execution order, into drafting, the **D-cut** selector, and verification; the drafter is shared, so the drafting bars are aligned and **D-cut**'s extra prefix-packing cost is folded into verification (which also covers sampling and bookkeeping). The selector adds only 0.55–0.58 ms per step (p95 $\leq$ 0.62 ms), which is 2.2–3.4% of the full step. In return, pruning shortens verification, the dominant term, by 23.7–38.3% and the full step by 21%

and 35% on GSM8K and MT-Bench. The selector cost is therefore small against the verification it removes, and the net effect is a shorter step.

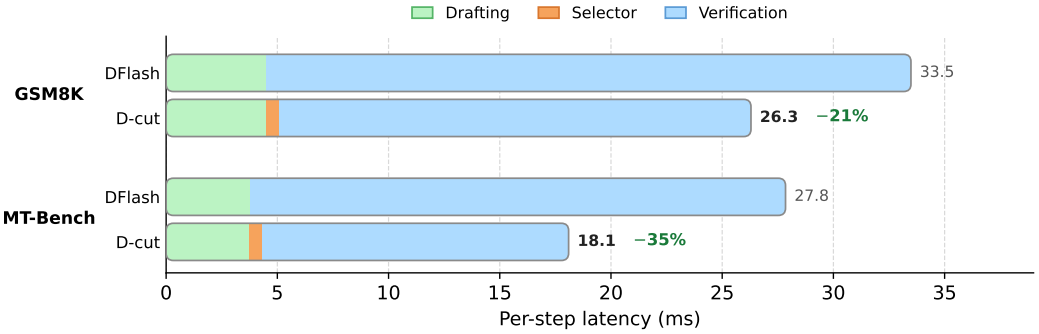


Figure 5: **Per-step latency at concurrency 64 (Qwen3-8B, H800).** Each bar splits the decoding step, in execution order, into drafting (green), the **D-cut** selector (orange), and verification (blue). The drafter is shared, so the drafting bars are aligned; **D-cut**'s extra prefix-packing cost is folded into verification, which also covers sampling and bookkeeping. Pruning shrinks verification, the dominant cost, and cuts the full step by 21% and 35% on GSM8K and MT-Bench, while the selector adds only 0.56–0.58 ms, about 2–3% of the step.

D Experimental Details

Draft models and checkpoints. Table 4 lists the draft model used for each baseline and target. The DFlash drafters are evaluated at verification block sizes of 8 and 16.

Table 4: **Draft models used for each baseline.** DFlash uses the official releases from https://github.com/z-lab/dflash for the public targets and the corresponding Hy3 checkpoint for Hy3-295B-A21B; EAGLE-3 uses the public AngelSlim heads; MTP reuses the multi-token-prediction head shipped with each target under its default configuration.

Method	Target	Draft checkpoint
DFlash	Llama-3.1-8B	`z-lab/LLaMA3.1-8B-Instruct-DFlash-UltraChat`
	Qwen3-4B	`z-lab/Qwen3-4B-DFlash-b16`
	Qwen3-8B	`z-lab/Qwen3-8B-DFlash-b16`
	Qwen3.5-27B	`z-lab/Qwen3.5-27B-DFlash`
	Qwen3.5-35B-A3B	`z-lab/Qwen3.5-35B-A3B-DFlash`
	Hy3-295B-A21B	`AngelSlim/Hy3-DFlash-b16`
EAGLE-3	Qwen3-4B	`AngelSlim/Qwen3-4B_eagle3`
	Qwen3-8B	`AngelSlim/Qwen3-8B_eagle3`
MTP	Qwen3.5-27B	built-in MTP head
	Qwen3.5-35B-A3B	built-in MTP head
	Hy3-295B-A21B	built-in MTP head

Serving configuration. All experiments run on a single node of 8× NVIDIA H20 GPUs, except the cross-hardware study in Section 4.4, which uses 8× NVIDIA H800 GPUs. The 8B-scale dense models run with tensor parallelism of 1, Qwen3.5-27B and Qwen3.5-35B-A3B use tensor parallelism of 4, and Hy3-295B-A21B uses tensor parallelism of 8. We build on vLLM and serve each model through its OpenAI-compatible chat endpoint, capping the running batch at 64 sequences (`max-num-seqs`) with up to 32,768 batched tokens per step, a maximum context length of 8,192, and 92% GPU memory utilization. We use the FlashAttention backend with piecewise CUDA-graph capture, and disable prefix caching and asynchronous scheduling to keep the measured throughput tied to the verification cost of each decoding step.

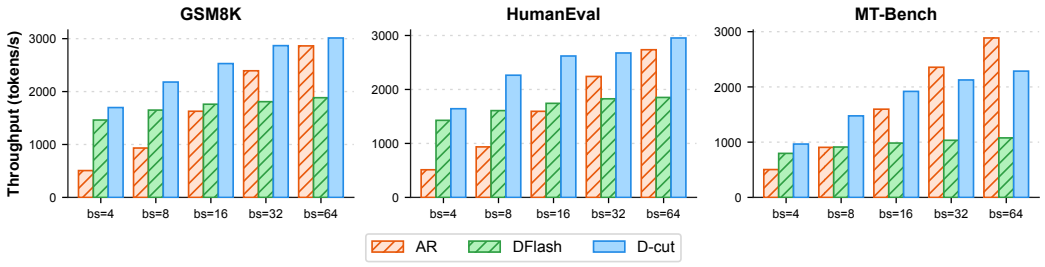


Figure 6: **Robustness under temperature-1 target sampling (Qwen3-8B, H20).** Throughput (tokens/s) as batch size grows from 4 to 64, with greedy draft proposals and target-only verification. Stochastic target sampling shortens accepted drafts, so DFlash gains little and is overtaken by AR decoding at high concurrency. **D-cut** (auto) prunes the now less useful suffix and stays faster than DFlash across the sweep, sustaining a $1.15\times$ average speedup over AR.

Benchmarks and decoding. For each benchmark, we use a fixed 256-prompt manifest and a maximum output length of 2,048 tokens, with greedy decoding (temperature 0); for the Qwen3 and Qwen3.5 models we disable the thinking mode. We throttle requests to the target concurrency level and warm up each serving configuration before recording throughput, so that the server operates at the intended batch size.

Hyperparameters. Table 5 collects the default configuration, so that the per-section text can stay focused on the variables it sweeps or overrides.

Table 5: **Default configuration used across experiments.** A few values are overridden by specific studies and stated in the corresponding section, such as the concurrency level and the decoding temperature.

Parameter	Value
Draft length D	15 (verification block of 16 positions)
Keep ratios $\mathcal{R}$	$\{0.25,\ 0.50,\ 0.75,\ 1.00\}$
Max running batch (`max-num-seqs`)	64
Max batched tokens per step	32,768
Max context length	8,192
GPU memory utilization	92%
Tensor parallelism	1 (dense 8B); 4 (Qwen3.5-27B, 35B-A3B); 8 (Hy3)
Attention backend	FlashAttention
CUDA graph	piecewise capture
Prefix caching / async scheduling	disabled
Decoding temperature	0 (greedy); 1 in Section 4.5
Thinking mode	off
Prompts per benchmark	256
Max output length	2,048
Concurrency sweep	$\{4,\ 8,\ 16,\ 32,\ 64\}$

Draft confidence predicts token acceptance. Figure 7 validates the confidence signal used by the selector without the selection bias introduced by **D-cut**'s pruning: we run DFlash-B16 with full verification on Qwen3-8B/H800, collect every draft position on GSM8K, HumanEval, and MT-Bench at concurrency 32 and 64, and label a position as accepted if it lies inside the verifier's accepted prefix. The prefix-product confidence $s_{i,k}$ is strongly monotonic with empirical acceptance across all three datasets: Spearman correlation is 0.748 on GSM8K, 0.770 on HumanEval, and 0.579 on MT-Bench, while AUROC is 0.957, 0.962, and 0.972, respectively. The goal of this signal is ranking rather than probability calibration, and the monotonic curves show that high-confidence positions are precisely the ones most worth preserving under a shared verification budget.

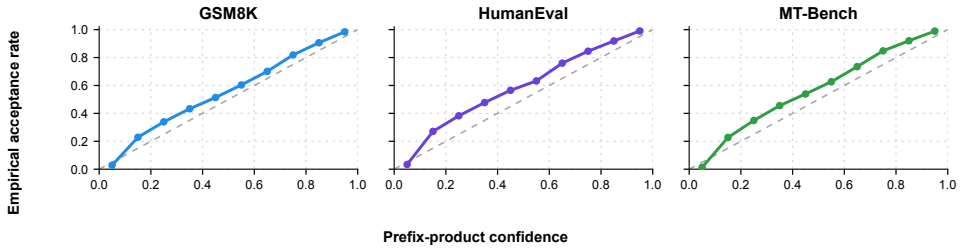


Figure 7: **Draft confidence versus verifier acceptance.** Token-level acceptance rate after binning DFlash-B16 draft positions by the prefix-product confidence $s_{i,k}$ on Qwen3-8B/H800, using full verification to avoid pruning-induced bias. The dashed gray line marks $y = x$ as a calibration reference. GSM8K, HumanEval, and MT-Bench all show a monotonic confidence-acceptance relationship, confirming that the same confidence used by **D-cut** is a strong ranking signal for accepted draft positions.

E Extended Related Work

Speculative decoding and candidate construction. Speculative decoding accelerates autoregressive generation through a lossless *draft-then-verify* paradigm, where a lightweight draft process proposes candidate tokens and the target model verifies them in parallel (Leviathan et al., 2023; Chen et al., 2023). Early methods use independent small models as drafters, while later methods improve candidate quality or diversity through tree verification and multi-head prediction. SpecInfer organizes candidates as token trees (Miao et al., 2024), Sequoia and OPT-Tree optimize tree structures under token or hardware budgets (Chen et al., 2025; Wang et al., 2024), and Medusa adds multiple decoding heads to the target model (Cai et al., 2024). Another branch reduces drafter-target mismatch by reusing target-model information: EAGLE predicts future hidden states from target representations (Li et al., 2025a), EAGLE-2/3 improve this line with dynamic trees and multi-layer features (Li et al., 2024; 2025b), GLIDE reuses the target KV cache (Du et al., 2024; Liu et al., 2026c), and multi-token prediction heads provide low-overhead self-drafting candidates (Gloeckle et al., 2024). DFlash further replaces autoregressive drafting with a block-parallel diffusion drafter conditioned on target features, generating an entire draft block in one forward pass (Chen et al., 2026). These works mainly improve how candidates are generated; **D-cut** instead focuses on which generated candidates should be verified in a batched step.

Static and dynamic verification budgets. Many speculative decoding methods implicitly use a fixed per-request draft shape, such as a fixed chain length or a fixed token tree (Liu et al., 2026a). Such static designs are simple and efficient, but they can waste verification compute on low-utility candidates when the drafter is uncertain. Adaptive methods address this by changing the draft length or tree online. SpecDec++ and DISCO adjust candidate length based on confidence or recent acceptance behavior (Huang et al., 2025; Mamou et al., 2024), while TALON, AdaEAGLE, and DySpec adapt draft-tree structures to the current context (Liu et al., 2026b; Zhang et al., 2024b; Xiong et al., 2024). Most prior adaptive methods reduce waste within a request, but their control objective is still centered on each request's local draft structure. In contrast, **D-cut** treats the batch as sharing a verification budget and reallocates this budget across requests according to relative draft confidence.

Speculative serving under high concurrency. Recent studies show that speculative decoding speedups can diminish under high concurrency, because batched verification must process bs $\times$ D draft tokens and can move the target model from a memory-bound regime to a compute-bound one (Liu et al., 2026d; Hu et al., 2026). PEARL addresses serving inefficiency by overlapping drafting and verification with adaptive draft length control (Liu et al., 2025). TETRIS studies cross-request allocation under a fixed verification capacity for autoregressive drafters (Wu et al., 2025). DSpark shares **D-cut**'s view of cross-request pruning and runtime-adaptive verification (Cheng et al., 2026). It uses a semi-autoregressive drafter and a trained confidence head, whereas **D-cut** reuses confidence from an existing block-parallel drafter without additional training. Most closely related to our setting, ECHO formulates high-concurrency speculative decoding as a budgeted scheduling problem and uses sparse confidence gating to coordinate tree depth and width under a global verification cap (Hu et al., 2026). **D-cut** shares the view that verification compute is the scarce resource in

batched serving, but targets a different drafter class: ECHO is designed for tree-structured or autoregressive drafters such as EAGLE and MTP, where the scheduler controls tree depth and width under a global verification cap. DFlash-like block-parallel drafters instead emit a single linear block per request, where the key decision is the verification depth retained for each request. **D-cut** therefore performs cross-request pruning over linear draft blocks and selects the pruning depth with a profiled runtime cost model, making the allocation adaptive to both batch confidence and deployment hardware.